ANL-FRA-165

ANL-FRA-165

Engineering Physics Production Code Implementation on the Cray X-MP

by

C. H. Adams, K. L. Derstine, and B. J. Toppel

Argonne National Laboratory, Argonne, Illinois 60439
Operated by the University of Chicago
for the United States Department of Energy Under Contract W-31-109-Eng-38

Base Technology

Argonne National Laboratory, with facilities in the states of Illinois and Idaho, is owned by the United States government, and operated by The University of Chicago under the provisions of a contract with the Department of Energy.

March 29, 1989 FRA-TM-165

ENGINEERING PHYSICS PRODUCTION CODE IMPLEMENTATION ON THE CRAY X-MP

by

C. H. Adams, K. L. Derstine, and B. J. Toppel

Engineering Physics Division
Argonne National Laboratory
Argonne, Illinois 60439

FRA TECHNICAL MEMORANDUM NO. 165

Work supported by the U.S. Department of Energy, Nuclear Energy Programs, under contract W-31-109-Eng-38.

ABSTRACT

Engineering Physics Production Code Implementation on the Cray X-MP

by

C. H. Adams, K. L. Derstine, and B. J. Toppel

This document describes Engineering Physics Division software designed to ease transition to the Cray X-MP computer and to provide support in the long term for a multi-computer environment for batch reactor analysis calculations. These activites include development of Unicos scripts to provide easier access to the new system, implementation of the File Transfer Utility FTU which provides format conversion of binary files between the Cray and other computer systems, and development of a modular environment analogous to the system which has been in use on the Argonne IBM computer system for many years.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

The arrival of a Cray X-MP/14 at Argonne significantly changed the computing environment for users of Engineering Physics Division (EP) reactor analysis programs. In a relatively short period of time the Central Computing Facility expanded from an IBM-only operation to one that offered batch and interactive computing on IBM, DEC and Cray machines. Engineering Physics Methods and Computational Support (M&C) Section undertook the development of several pieces of software designed not only to make the transition easier but also to provide support in the long term for a multi-computer environment for batch reactor analysis calculations.

The UNICOS operating system on the Cray is quite different from the IBM Job-Control Language (JCL) that EP users were accustomed to. Most users did not want to learn Unix to the degree necessary to maintain the job-control statements in their batch-job input files. Chapter 2 describes APSHELL, a system of UNICOS shell variables and scripts which provides an easy-to-use environment for the kind of batch work that makes up most of the Division's computing. The M&C Section will maintain APSHELL in such a way as to accomodate future system changes without requiring users to modify their job input. When the Division acquires Unix-based workstations we intend to adapt APSHELL to them to provide continuity.

Ancillary calculations are often performed following a long production calculation making use of smaller codes on the IBM computers at ANL. With the production codes moving over to the Cray computer, there was a need to provide a binary file format converter so that binary files written on the Cray could be read on the IBM computers, and vice versa. Chapter 3 describes the File Transfer Utility, FTU which permits communication via binary files between the Cray, IBM, VAX, and CDC computers.

The advantages of modular programming have been enjoyed at ANL since the development of the ARC System [1]. Since the UNICOS operating system does not provide the capability of linking modular codes at run time, this effort has included the development of a modular programming system on the Cray, IBM, and VAX computer systems which essentially duplicates the capabilities afforded by the IBM operating system and is far more portable. Chapter 4 describes the characteristics of this modular programming system.

A number of places in this document refer to procedures or scripts with the prefix AP rather than EP (e.g. APSHELL, APPROC, etc.). This is done for historical reasons since their development occurred in the Applied Physics Division before it was combined with the Engineering Division and renamed the Engineering Physics Division.

Chapter 2

APSHELL, UNICOS SCRIPTS FOR BATCH COMPUTING

2.1 *Introduction*

This set of UNICOS shell variables and scripts is intended to provide a batch computing environment in which users of the Engineering Physics Division's reactor analysis codes can work conveniently and safely without having to write or maintain a lot of UNICOS JCL themselves. In some sense, they serve the same sort of function that the catalogue procedures ARCSP0nn have been providing on the IBM machines. However, UNICOS and IBM/MVS are entirely different kinds of operating systems, and direct comparisons between the ARCSP procedures and APSHELL are generally not useful. APSHELL was developed with the following objectives in mind:

> EP's production codes run in a batch environment, but UNICOS is fundamentally an interactive system. Batch jobs, running unattended, need simple-to-use tools for making basic decisions about how to handle errors.
>
> Scripts that fetch and dispose files have built-in, default IBM JCL, but since EP's IBM users generally feel comfortable about overwriting DD cards in IBM catalogued procedures, those scripts permit the entry of IBM JCL to change the default JCL of files.
>
> Users are permitted, and even encouraged, to mix UNICOS and APSHELL commands.
>
> UNICOS system output is just as eye-tangling to read as the IBM's JESMSG, JESJCL and SYSMSG files. APSHELL provides easily-spotted error messages.
>
> To the extent possible, APSHELL provides English-language information and error messages.
>
> The APSHELL syntax is different from UNICOS; in particular APSHELL commands are in upper case and do not follow the UNICOS practice of identifying parameters with hyphens. This is done partly to simplify the syntax and partly to make a clear distinction between UNICOS and APSHELL commands.

For the kind of work we do in EP, APSHELL lets users set up jobs with little or no IBM JCL and with a minimum of Cray commands. The APSHELL scripts perform some of the basic functions needed in batch computing on Argonne's Cray X-MP:

> `STAGEIN` and `STAGEOUT` manage the flow of files between the X-MP and the IBM disk farm. Although they are built around the UNICOS commands "fetch" and "dispose", they include error checking and offer built-in, default IBM JCL.
>
> `PREPATH` and `POSTPATH` modify the search path to include access to additional directories containing other scripts.
>
> `PRELIB` and `POSTLIB` make available libraries of load modules.

IFBOMB and STASHFILE are tools to be used in detecting errors and recovering from various kinds of abnormal terminations.

APSHELL is an initialization script.

A number of additional, relatively simple scripts (XDIF3D, XFTU, XREBUS3, XTWODANT, XVIM, XBANDIT, XRETALLY, and XVARI3D) are available for some production codes.

Beside these scripts, the APSHELL environment includes a number of shell variables which users can set to simplify the disposition of output and the definition of job parameters.

Figure 1 illustrates a very simple DIF3D [2] job that uses APSHELL shell variables and scripts. The job fetches a Cray-format ISOTXS [3] data set and, if the fetch was successful, executes DIF3D. The DIF3D output and the standard-output file are sent to the 3800 printer.

```
# user=
# QSUB-r CHA01
# QSUB -q y
# QSUB -lT 80
# QSUB -lM 500kw
# QSUB -eo
# QSUB
APOUTPUT=output
APDEST=PR0
. APSHELL
cat > input <<'EOD'
BLOCK=OLD
DATASET=ISOTXS
BLOCK=STP021
        .
        .
  BCD job input
        .
        .
EOD
STAGEIN CRAYBINARY ISOTXS B21006.VERY.TEMP.CRAY.ISOTXS
IFBOMB
XDIF3D
```

Figure 1. Simple DIF3D Job Using APSHELL Shell Variables

Figure 2 shows a much more complicated job that fetches a number of different kinds of files from the IBM disks, compiles and links a UDOIT into a stand-alone DIF3D, uses the FTU utility (Chapter 3) to convert an ISOTXS file from IBM to Cray format, executes DIF3D and finally disposes an RTFLUX [3]

file and the Cray load module back to the IBM disks, a copy of the output to microfiche and a PostScript graphics file to the LaserWriter at RADS12. There are a number of error checks along the way to abort the job if a problem arises.

```
# user=
# QSUB -r CHA06
# QSUB -q y
# QSUB -lT 70
# QSUB -lM 1000Kw
# QSUB -eo
# QSUB
set -vx
APOUTPUT=output
APDEST=PR0,FLASH=LINE
APLASERWRITER=RM112PR2
APBANNER=sample2
APFETCH=YES
. APSHELL
# ######################################################################
#   COMPILE A UDOIT MODULE AND LINK INTO DIF3D.
# ######################################################################
STAGEIN CARDS program.f B21006.VERY.TEMP.FORTRAN 'DISP=(OLD,DELETE)'
cft77 -e Dcxs program.f >> output
IFBOMB banner 'cft77' >> output
grep 'error' program.l >> output
grep 'warning' program.l >> output
cat program.l >> output
cat <<END_OF_FILE > directs
LIB=/n1/b21006/cccc/seglib.a,/n1/b21006/cccc/syslib.a
LIB=/n1/b21006/cccc/calanl.a,$ANLUTIL,$DISLIB
ECHO=ON
MODULES=ENDFDA:/n1/b21006/cccc/seglib.a
USX=WARNING
MAP=ALPHA
FORCE=ON
ABS=dif3d
END_OF_FILE
segldr program.o directs >> output
IFBOMB
# ######################################################################
#   STAGEIN ISOTXS, EXECUTE FTU.
# ######################################################################
STAGEIN IBMBINARY STACK B21006.VERY.TEMP.IBM.ISOTXS
cat > input <<'EOF'
ISOTXS 1 CRAY IBM
EOF
```

Figure 2. Elaborate Example of the Use of the APSHELL Shell Variables

```
XFTU
rm STACK
IFBOMB
# ######################################################################
#   EXECUTE DIF3D WITH THE UDOIT.
# ######################################################################
cat <<EOD> input
BLOCK=OLD
DATASET=ISOTXS
BLOCK=STP021
       .
       .
  BCD job input
       .
       .
EOD
XDIF3D
STAGEOUT POSTSCRIPT fort.98
STAGEOUT OUTPUT output '(ANLVM,FICHE)'
IFBOMB
STAGEOUT LOADMODULE dif3d B21006.CRAY.UDOIT.MODLIB    \
 'DISP=(NEW,CATLG),UNIT=TEMP,SPACE=(TRK,(60,5),RLSE)'          \
 'DCB'
```

Figure 2. Elaborate Example of the Use of the APSHELL Shell Variables (cont'd.)

The next section of this writeup describes a set of shell variables which the user may want to set. The last section describes the shell scripts available and shows examples of their use.

2.2 User Shell Variables

Users may set default definitions for a number of job parameters by means of the "shell variables" described in this section. In most cases they should probably be defined before the initialization call to APSHELL so that they are available to all of the APSHELL scripts invoked during a job, but there are situations where users will want to change them during execution. For most of the following shell variables the initialization call to APSHELL will set default definitions if the user has not supplied them.

2.2.1 APBANNER

APBANNER contains a string (up to 10 characters) that will be printed at the top of the header page in big letters.

If the shell variable APBANNER is to be set, it must be set before the initializing command APSHELL.

APBANNER is used by the APHEADER script.

Example: `APBANNER='hi there'`

Default: the NQS job name (the "request-name" define by `QSUB -r`)

2.2.2 APDEST

This is the default destination for standard output and other printed output. The destination can include other parameters recognized by the MVS Station; one of the examples below turns on the forms flash for the 3800 printer. Also see the examples below for the special format required to make the default destination a CMS session.

To be fully effective this parameter should be set before the initialization call to `APSHELL`.

`APDEST` is used by `APSTART` and `APWRAP`.

Examples:
```
APDEST=RADS12
APDEST=PRO,FLASH=LINE
APDEST=PRO,COPIES=2
APDEST='(ANLVM,B21006)'
APDEST=VMFICHE
```

Default: null

2.2.3 APFETCH

When `APFETCH=YES` both the standard output file and the output file defined by the `APOUTPUT` shell variable will be routed to Wylbur FETCH before printing, regardless of what is specified in the IBM JCL that submits the Cray job. If this shell variable is set to anything else the MVS station determines whether or not the output is to be FETCHed from the IBM JCL that submits the job.

If the shell variable `APFETCH` is to be set, it should be set before the initializing command `APSHELL`.

`APFETCH` is used by the `APSTART` and `APWRAP` scripts.

Examples:
```
APFETCH=YES
APFETCH=NO
```

Default: null

When `APFETCH=YES` is used with `APDEST=VMFICHE`, the output files can be `FETCH`ed before they are sent to the fiche unit. When `APDEST='(ANLVM,FICHE)'` the output files go directly to the fiche unit and cannot be `FETCH`ed.

2.2.4 APINPUT

`APINPUT` is the default BCD input file name for the production-code scripts `XDIF3D`, `XFTU`, `XREBUS3`, `XTWODANT`, `XVIM`, `XBANDIT`, `XRETALLY`, and `XVARI3D`. Those scripts also permit the user to specify the BCD input file name as an argument.

The shell variable APINPUT can be set at any time.

APINPUT is used by the XDIF3D, XFTU, XREBUS3, XTWODANT, XVIM, XBANDIT, XRETALLY, and XVARI3D. scripts.

Examples: APINPUT=my_input

Default: input

2.2.5 APLASERWRITER

APLASERWRITER is the default destination for PostScript files handled via the STAGEOUT script.

The shell variable APLASERWRITER can be set any time before the command STAGEOUT.

Examples: APLASERWRITER=RM112PR2 (RADS12)
APLASERWRITER=RM010PR3 (ZPPR)

Default: RM113PR2 (building 221)

2.2.6 APLINES

APLINES contains the file line limit for the job (i.e. the JCL parameter OUTLIM). Supply the limit in actual number of lines (not in thousands as on the IBM machine).

If the shell variable APLINES is to be set, it should be set before the initializing command APSHELL.

APLINES is used by the APSTART, STAGEOUT and APWRAP scripts.

Examples: APLINES=100000

Default: 10000

2.2.7 APMESSAGES

Normally the APSHELL scripts print only important messages; individual commands within each script are not echoed as they are executed. When APMESSAGES=VERBOSE all commands executed within each script are echoed. This can produce a very busy standard output file, but it is sometimes useful in debugging a script.

APMESSAGES can be redefined as often as necessary during an execution to toggle the command echoing.

APMESSAGES is used by all the APSHELL scripts.

Examples: APMESSAGES=VERBOSE
APMESSAGES=

Default: null

2.2.8 APOUTPUT

This is the name of a file to which carriage-controlled output is to be written. The user is responsible for directing such output to the file when he issues UNICOS commands; the APSHELL scripts XDIF3D, XFTU, XREBUS3, etc. automatically route printed output from production codes to that file. At the completion of the job the file will automatically be directed to the destination specified by the APDEST shell variable, subject to how the user has set APFETCH.

To be fully effective this parameter should be set before the initialization call to APSHELL.

APOUTPUT is used by APSTART, APWRAP, and the production-code scripts XDIF3D, XVARI3D, etc.

Examples:
```
APOUTPUT=output
APOUTPUT=fileout
```

Default: carriage-controlled output is written to the standard output file.

2.2.9 APPOSTPATH

APPOSTPATH defines extensions to the search path that are to be concatenated at the end of the search path and that are to remain there for the duration of the job.

If the shell variable APPOSTPATH is to be set, it must be set before the initializing command APSHELL. To add *additional* paths (and then to remove them) during execution use the POSTPATH script.

APPOSTPATH is used by the APSTART and POSTPATH scripts.

Examples:
```
APPOSTPATH=$HOME/my_scripts
APPOSTPATH=$HOME/my_scripts:/n1/bnnnnn/his_scripts
```

Default: null

2.2.10 APPREPATH

APPREPATH defines extensions to the search path that are to be concatenated at the front of the search path and that are to remain there for the duration of the job.

If the shell variable APPREPATH is to be set, it must be set before the initializing command APSHELL. To add *additional* paths (and then to remove them) during execution use the PREPATH script.

APPREPATH is used by the APSTART and PREPATH scripts.

Examples:
```
APPREPATH=$HOME/my_scripts
APPREPATH=$HOME/my_scripts:/n1/bnnnnn/his_scripts
```

Default: null

2.2.11 APTEST

APTEST is a shell variable used by the APSHELL scripts as an error-condition sentinel for the IFBOMB script. Users may occasionally want to clear a previous error condition by setting this variable to null.

APTEST is read by the IFBOMB script but may be set by any of the APSHELL scripts.

Example: APTEST= (to clear error-condition sentinel)

Default: null

2.3 User APSHELL Scripts

A number of scripts have been written which are of general interest to both the shell programmer and to users. The latter will find these advantageous since they largely eliminate the need for extensive knowledge of the UNIX command language. In order to use any of these scripts, the APSHELL script must be executed first, and the command that executes it must be preceded by a "dot-space" (see below). None of the other commands require the "dot-space".

2.3.1 . APSHELL

This is an initialization call. It *must* come first, or none of the other scripts will be available to the job. Users should define most of the shell variables described in the previous section before invoking APSHELL.

This is a one-line script residing in the public directory /usr/public. It, in turn, executes the script APSTART which, among other functions, sets default values for the shell variables defined in the previous section, changes to a SCRATCH directory, starts the accounting utility, modifies the search path to make other APSHELL scripts accessible, sets a trap that causes the script APWRAP to be executed when the job terminates, invokes uscproute, and invokes the APHEADER script to print the header page showing shell variable definitions. The APWRAP script prints an accounting report (including a breakdown of costs by charge category provided by the script /n1/b27484/jobcost) and disposes the output file defined by APOUTPUT.

There are no arguments.

2.3.2 APECHO "string1" "string2" "string3" ...

This script echos the input strings beginning each string on a new line. There may be any number of strings supplied as arguments. The strings are echoed both to the standard output, and to APOUTPUT if it exists.

Examples:
```
APECHO "This is line 1" "This is line 2"
APECHO " "   " "   " "
```

If no arguments are provided, APECHO does nothing.

2.3.3 IFBOMB command

This script checks to see if there was an error either in the execution of the immediately preceding UNICOS command or in the execution of any earlier APSHELL script. If there was no error, the script does nothing. If an error is detected, and if there *is no argument*, the script invokes the APWRAP script and the job terminates. If an error is detected, and there *is an argument*, IFBOMB executes the argument as a command and lets the job continue.

Errors in the immediately preceding UNICOS command are detected by testing the shell variable $?; if an error is detected this way the shell variable APTEST is set equal to "previous command". Errors from earlier APSHELL scripts are caught by testing APTEST. At any time the user may reset APTEST to null to avoid detonating later IFBOMBs.

The argument can be any UNICOS or APSHELL command.

Examples:

```
IFBOMB
IFBOMB STASHFILE STACK
IFBOMB banner 'hey stupid'
IFBOMB banner 'file gone' >> output
IFBOMB APTEST=
IFBOMB STAGEIN CARDS file1 B21006.INPUT
IFBOMB STAGEIN CARDS file2 B21006.INPUT \
          "'DISP=(OLD,DELETE)'"
IFBOMB echo first IFBOMB test
```

Note the use of single and double quotes in the arguments of some of the examples. The syntax of commands whose arguments include commands with arguments is not always straightforward.

2.3.4 POSTLIB ibmfile1 ibmfile2 .... ibmfilen

This script permits users to supplement the default production load module library with modules from user-specified libraries. It acquires UNICOS ar module libraries from the IBM disks and places them in the directory $SCRATCH/xpostlib that is added to the end of the search path. Modules in earlier named ibmfiles preempt modules in succeeding ibmfiles. Modules acquired with additional POSTLIB commands (if any) will replace identically named modules existing in $SCRATCH/xpostlib, but the position of $SCRATCH/xpostlib will not be changed in the search path. If there are no arguments, the script removes all modules from the xpostlib directory.

This script along with PRELIB, PREPATH and POSTPATH change the search path when they are called. Specifying POSTLIB with no arguments empties the xpostlib directory, but does not remove the directory from the search path. Specifying PREPATH or POSTPATH with no arguments will restore the search path to its original state. The original search path includes a directory /n1/b05432/cccc_modlib containing production load modules.

Example:

```
POSTLIB  C116.TEST.POSTLIB1   C116.TEST.POSTLIB2
```

or equivalently,

```
POSTLIB  C116.CRAY.TEST.POSTLIB2
POSTLIB  C116.CRAY.TEST.POSTLIB1
```

2.3.5 POSTPATH path1 path2 .... pathn

This script permits users to modify the search path during execution by adding directories at the end. Any number of arguments is allowed. If there are no arguments, the script removes path names appended by earlier calls (but does not remove paths defined by the shell variables APPREPATH and APPOSTPATH) The original search path includes a directory /n1/b05432/cccc_modlib containing production load modules.

The scripts PREPATH, PRELIB, and POSTLIB also change the search path.

Example: `POSTPATH /n1/b54321/debug /n1/b12345/other`

2.3.6 PRELIB ibmfile1 ibmfile2 .... ibmfilen

This script permits users to preempt the default production load module directory with modules from user-specified libraries. It acquires UNICOS ar module libraries from the IBM disks and places them in the directory `$SCRATCH/xprelib` that is added to the front of the search path. Modules in earlier named ibmfiles preempt modules in succeeding ibmfiles. Modules acquired with additional `PRELIB` commands (if any) will replace identically named modules existing in `$SCRATCH/xprelib`, but the position of `$SCRATCH/xprelib` will not be changed in the search path. If there are no arguments, the script removes all modules from the `xprelib` directory.

This script along with `POSTLIB`, `PREPATH` and `POSTPATH` change the search path when they are called. Specifying `PRELIB` with no arguments empties the `xprelib` directory, but does not remove the directory from the search path. Specifying `PREPATH` or `POSTPATH` with no arguments will restore the search path to its original state.

Example:
```
PRELIB  C116.TEST.PRELIB1   C116.TEST.PRELIB2
```

or equivalently,

```
PRELIB  C116.CRAY.TEST.PRELIB2
PRELIB  C116.CRAY.TEST.PRELIB1
```

2.3.7 PREPATH path1 path2 .... pathn

This script permits users to modify the search path during execution by adding directories at the front. Any number of arguments is allowed. If there are no arguments, the script removes path names appended by earlier calls (but does not remove paths defined by the shell variables APPREPATH and APPOSTPATH) The original search path includes a directory /n1/b05432/cccc_modlib containing production load modules.

The scripts POSTPATH, PRELIB, and POSTLIB also change the search path.

Examples: `PREPATH /n1/b54321/debug /n1/b12345/other`

2.3.8 STAGEIN type crayfile ibmfile jcl jcl jcl

If the file `crayfile` does not already exist, this script brings it to the Cray from the IBM disks. `STAGEIN` basically performs the same function as the UNICOS "acquire" command. The user specifies the type of file via a `type` keyword in the first argument and the Cray and IBM file names in the second and third arguments. The fourth, fifth, and sixth arguments are optional patches of JCL; if there are no JCL arguments, the script assumes that the IBM file is `DISP=(OLD,KEEP)`.

If `crayfile` already exists `STAGEIN` does nothing. If `crayfile` does not already exist, and the first attempt to fetch a file fails, the script tries a second time. When `type=LOADMODULE` the script marks the load module executable.

When `type=MODLIB` the script extracts all members from an ar library into the directory specified in `crayfile`. If the specified directory does not exist, `STAGEIN` will create it.

`type` One of the following choices:

`CARDS`	for BCD card-image data sets.
`SEGLIB`	for relocatable object code.
`CRAYBINARY`	for Cray-binary-format data sets.
`IBMBINARY`	for IBM-binary-format data sets.
`LOADMODULE`	for executable load modules.
`ARLIB`	for ar load module libraries.
`MODLIB`	for extracting all members from an ar library into the directory specified in crayfile

`crayfile` The name of the file on the Cray, or a directory name if `type=MODLIB`.

`ibmfile` The name of the dataset on the IBM disks.

`jcl` Optional JCL for the IBM file. The default is `DISP=(OLD,KEEP)` if no JCL input is provided.

Examples:

```
STAGEIN CARDS program.f B21006.FORTRAN
STAGEIN IBMBINARY data B21006.BINARY.DATA   \
      '(OLD,DELETE)'
STAGEIN CARDS input 'B21006.DECKS(GAMSORA)'
STAGEIN SEGLIB seglib.a B21006.CRAY.SEGLIB
STAGEIN MODLIB $SCRATCH/xprelib B21006.CRAY.SEGLIB
STAGEIN LOADMODULE ftu C116.CRAY.FTU.MODLIB
STAGEIN IBMBINARY STACK B21006.TEST.ISOTXS
STAGEIN SEGLIB seglib.a B21006.TEST.SEGLIB  \
      'DISP=OLD' 'VOL=SER=TEM401' 'UNIT=TEMP'
STAGEIN SEGLIB seglib.a B21006.TEST.SEGLIB 'DISP=OLD,\
VOL=SER=TEM401,UNIT=TEMP'
```

Note that the three JCL patches may be presented in any form which is recognizable as Unix strings. The last two examples illustrate two ways that the same information can be presented. The last example provides all the JCL information in one string whereas the next to the last example uses three JCL input strings.

2.3.9 STAGEOUT type crayfile ibmfile jcl jcl jcl

This script sends a variety of types of files from the Cray to the IBM disks for storage or to hard-copy output devices. The user specifies the type of file via a `type` keyword in the first argument and the Cray and IBM file names in the second and third arguments. The fourth, fifth and sixth arguments are optional patches of JCL; if there are only three arguments the script assumes that disk files are `DISP=(OLD,KEEP)`. When `type` is `OUTPUT` or `POSTSCRIPT` the third argument is interpreted as the destination (see the examples below); in these two cases if the third argument is omitted the destination is picked up from `APDEST` or `APLASERWRITER`, respectively.

If a previously executed APSHELL script has set an error condition (i.e. if the shell variable `APTEST` is not null) `STAGEOUT` will not send a file to the IBM side of the system. Users can override this behavior by setting `APTEST` to null before invoking STAGEOUT.

For `(NEW,CATLG)` files users *must* supply all the IBM JCL required to establish the file on the IBM disks. The one exception to this rule is the DCB parameter; if one of the `jcl` patches contains only the string `DCB,` then `STAGEOUT` will supply appropriate, default DCB parameters (see the examples below).

`type` One of the following choices:

`CARDS`	for BCD card-image data sets.
`SEGLIB`	for relocatable object code.
`CRAYBINARY`	for Cray-binary-format data sets.
`IBMBINARY`	for IBM-binary-format data sets.
`LOADMODULE`	for executable load modules.
`ARLIB`	for ar load module libraries.
`POSTSCRIPT`	for LaserWriter output.
`METAFILE`	for Issco metafiles.
`OUTPUT`	for printed output or microfiche.

`crayfile` The name of the file on the Cray.

`ibmfile` The name of the dataset on the IBM disks.

`jcl` Optional JCL for the IBM file. The default is `(OLD,KEEP)`.

Examples:

```
STAGEOUT CARDS program.f B21006.FORTRAN
STAGEOUT IBMBINARY data B21006.BINARY.DATA
STAGEOUT POSTSCRIPT fort.98
STAGEOUT OUTPUT output '(ANLVM,FICHE)'
STAGEOUT OUTPUT output VMFICHE
STAGEOUT LOADMODULE dif3d B21006.UDOIT.MODLIB \
      'DISP=(NEW,CATLG),UNIT=TEMP'  \
      'SPACE=(TRK,(60,5),RLSE)'  \
      'DCB'
STAGEOUT METAFILE popfil B21006.CHA09.POPFIL  \
      'DISP=(NEW,CATLG),UNIT=TEMP' \
      'SPACE=(TRK,(10,10),RLSE)' \
      'DCB'
STAGEOUT CARDS fort.7 B21006.CHA09.PUNFIL  \
      'DISP=(NEW,CATLG),UNIT=TEMP,SPACE=(TRK,(5,1),RLSE)' \
      'DCB'
```

```
STAGEOUT CARDS file2 B21006.INPUT \
      'DISP=(NEW,CATLG),UNIT=TEMP' \
      'SPACE=(TRK,(1,1),RLSE)' \
      'DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)'
STAGEOUT OUTPUT output '(ANLVM,B21006)'
STAGEOUT OUTPUT output RADS12
STAGEOUT OUTPUT output PR0,FLASH=LINE
```

Note that when one uses STAGEOUT to an ANLVM node (e.g. (ANLVM,B21006) or (ANLVM,FICHE)), it is not possible to make the output fetchable with `APFETCH=YES`. `APFETCH=YES` does operate on files sent to ANLOS (e.g. RADS12, PR0, and VMFICHE).

2.3.10 STASHFILE crayfile

This script will create a temporary directory with a name unique to the current job (`$SHORT/stash/nnnn.xmp`, where `nnnn` is the job's NQS job number) and will save a file `crayfile` in that directory. Multiple calls to `STASHFILE` will save additional files, all in the same directory. All $SHORT files will remain on the Cray disks for 24 hours from their last access. A line on the APSHELL header page tells users whether or not they have a "stash" directory created in a previous job. `STASHFILE` can be used with `IFBOMB` to save valuable files in case of trouble (see the example below).

Examples:
```
STASHFILE RTFLUX
IFBOMB STASHFILE STACK
```

2.3.11 XDIF3D crayfile, XFTU crayfile, XREBUS3 crayfile, XTWODANT crayfile, XVIM crayfile, XVARI3D crayfile

The scripts `XDIF3D crayfile`, `XFTU crayfile`, `XREBUS3 crayfile`, `XTWODANT crayfile`, and `XVARI3D crayfile` normally execute the production versions of the appropriate code. The single argument defines the file containing the BCD input for the code; if there is no argument the code will take its input from the file defined by `APINPUT`. The output of the code is directed to the file defined by `APOUTPUT`. Unix style output redirection must not be done on modular scripts as discussed in Chapter 4. Auxiliary output may also be created on the file fort.10. Symbolic dump output from abnormal terminations is generated on the file sysudump. In case of an abnormal termination, some of these scripts execute `STASHFILE` for the potentially available files:

Code	*Files Saved*
DIF3D	RTFLUX, ATFLUX (version 1)
REBUS3	RFILES,STACK
TWODANT	RTFLUX, ATFLUX (version 1)
VARI3D	RTFLUX, ATFLUX (versions 1-3)
VIM	none
RETALLY	none
BANDIT	none
FTU	none

The search path always includes the directory /n1/b05432/cccc_modlib containing all modules from the production library. Scripts PRELIB, POSTLIB, PREPATH, and POSTPATH may be

used to modify the search path prior to a call to one of the module execution scripts described in this section.

2.4 Shell Programmer Shell Varibles

A number of "shell variables" are available which are primarily of interest to the shell programmer.

2.4.1 APTEMP, APTEMP2, APTEMPX, APTEMPS, APTEMPS2, APTEMPS5, etc.

These are temporary variables used within many of the `APSHELL` scripts. They are not intended to be used to communicate between scripts. Programmers writing scripts that invoke APSHELL scripts should take some care in defining temporary variables. For example, the production-code scripts (e.g. `XDIF3D`, `XFTU`, etc.) use a temporary variable named `APTEMPX` so as not to conflict with the `APTEMPS` variable that is set if they invoke `STAGEIN`.

2.5 Shell Programmer APSHELL Scripts

A number of scripts have been written which are primarily of interest to the shell programmer and which are used in conjunction with scripts used for execution of the EP production codes such as DIF3D, REBUS3, etc.

2.5.1 APHEADER

This script collects and prints the header information to standard output and to the file identified by `APOUTPUT`. The header information includes job name and numbers, user-specified time and memory limits, the contents of a number of shell variables, and a list of APSHELL scripts. `APHEADER` is invoked by the `APSTART` script and has no arguments.

2.5.2 APSTART file

This script is invoked by the `APSHELL` script residing in the public directory `/usr/public`. `APSTART` changes directory to $SCRATCH, sets the initial values for a number of APSHELL shell variables, adds the directories /n1/b21006/apshell and /n1/b05432/cccc_modlib to the search path, issues the scpreroute (uscproute under UNICOS 4.0), starts the accounting deamon (jad in UNICOS 3.0, ja in UNICOS 4.0), invokes `APHEADER` to display the header page on the output files, sets a trap so that `APWRAP` is executed on exits, defines shell functions for all the APSHELL scripts to assure that they are all executed in the same shell, and lists the contents of the user's /n1 and /s1 directories and subdirectories. The optional argument is the name of any script that the programmer wishes to be executed at the end of `APSTART`.

2.5.3 APWRAP code

This script is invoked automatically on exit because of a trap set in `APSTART`. It prints accounting information to standard output and to the file identified by `APOUTPUT`, disposes any `APOUTPUT` file to its intended destination, and cleans out the $SCRATCH directory. The optional argument is a condition code that is used as the argument in a final call to exit.

Chapter 3

THE FILE TRANSFER UTILITY, FTU

3.1 Introduction

The availability of onsite Cray, IBM, and VAX computers as well as telecommunication access to offsite CDC and Cray computers has created an atmosphere in which users are more and more interested in making use of binary data files created on one computer as input for codes which are to be executed on another computer. Common examples are the use of post-processing codes which massage data generated by a large production-type calculation, or the use of cross section files generated on one computer with production codes to be run on another computer. Post-processing codes have been used on the IBM computers for many years. Users who now are running large production jobs on the more cost effective Cray computer are often not interested in expending the effort to convert the ancillary codes for use on the Cray, but would rather prefer to continue running the inexpensive post-processing job on the IBM computer using the binary files which were written on the Cray computer. ISOTXS microsopic cross section datasets [3] which have been generated on the IBM computers are now required on the Cray computer for a number of production codes. Similarly, in the future since the MC^2–2 code [4] is being implemented on the VAX computer, VAX generated ISOTXS datasets may also be used for the Cray production jobs. These considerations have motivated the development of the File Transfer Utility, FTU, which will afford portability for a variety of binary datasets from one type of computer to another.

3.2 FTU User Information

FTU, programmed in Fortran 77, is intended to provide binary file format conversion between the Cray and the IBM, VAX, and CDC computers[1], as well as the ability to copy datasets, without format conversion, from a file on which a number of datasets have been "stacked". This latter capability is similar to that provided by the READSTAK code [5] on the IBM computers which is employed to copy datatsets from a so-called STACK file which has been written during a REBUS-3 [6] job. In addition to the new format conversion capability, FTU processes 11 additional datasets not addressed by READSTAK, and can provide BCD equivalents and edits for each of the datasets. The BCD format affords easy portability between computer systems, and the dataset edits provide a useful debugging tool for code development activities or for users who would like to verify the contents of a binary dataset.

3.2.1 Files Processed by FTU

Table 1 lists the datasets which are currently processed by FTU. The datasets DLAYXS, FIXSRC, GEODST, ISOTXS, NDXSRF, PWDINT, RAFLUX, RTFLUX, RZFLUX, AND ZNATDN are standard CCCC interface files [3]. The emphasized datasets are those which are processed by the READSTAK

---

1 FTU has not as yet addressed format conversion for CDC format datasets nor conversion from BCD to binary format.

code.

Table 1. Datasets Processed by FTU

AAFLUX	ADJANG	ANGSRC	ATFLUX	COMPXS	DIRANG
DISFAC	DLAYXS	D3EDIT	FIXSRC	**GEODST**	ISOTXS
LABELS	NAFLUX	**NDXSRF**	**NHFLUX**	PMATRX	**PWDINT**
RAFLUX	**RTFLUX**	**RZFLUX**	SFEDIT	**ZNATDN**	

3.2.2 FTU Input Conventions

The input to FTU is specified using free field format style and contains

file_name record_number output_format input_format precision

where *file_name* corresponds to one of the datasets in Table 1, *record_number* indicates the record on the STACK file where the file is located, *output_format* may be CRAY, IBM, VAX, CDC, or BCD corresponding to the binary file formats on the Cray, IBM, VAX or CDC computers, or to BCD format, *input_format* may be CRAY, IBM, VAX, or CDC, again corresponding to the file formats on those computers, and *precision* is SINGLE or DOUBLE correpsonding to 4-byte or 8-byte word length for conversion from Cray to IBM. As implemented on the Cray, *output_format* is defaulted to IBM and *input_format* is defaulted to CRAY. The input sentinel *precision* is pertinent at this time only for the FIXSRC dataset which is required in both single- and double-precision versions on the IBM computers. If this field is left blank, single precision conversion is assumed consistent with the CCCC specification for that dataset. [3]. The file containing dataset *file_name* is always given the name STACK whether or not it contains more than one of the datasets in Table 1.

On the Cray computer, if *input_format* is other than CRAY, then *output_format* must be CRAY. When implemented on computers other than the Cray, FTU only provides the capability of copying the datasets from a STACK file, without format conversion, or generating the equivalent BCD form for the datasets. Thus, for example on the IBM computer, *input_format* may only be IBM and *output_format* may only be IBM or BCD. Note that FTU may be used to convert IBM to VAX format, e.g., by first converting from IBM to Cray format, and then from Cray to VAX format using the Cray implementation of FTU.

If *record_number* is supplied as negative, FTU will provide an edit of the file being created. The dataset edits as well as the structure of the BCD equivalents for the datasets are quite terse so that the user will have to be familiar with a description of the file being processed in order to follow the FTU edit. The datasets specified by the CCCC are documented in Ref. [3]. The other datasets in Table 1 which are processed by FTU are documented in the Wylbur accessible partitioned dataset B21006.FILES#filename where filename corresponds to any of the datasets in Table 1. When converting individual datasets (that is when the STACK file contains only one dataset), *record_number* would be 1 or -1.

3.2.3 Cray Execution of FTU

Figure 3 shows a typical Cray execution of FTU in which an IBM format RTFLUX dataset [3] is converted to a Cray format version, and the new Cray dataset is then converted to BCD equivalent format. The first step also provides an edit of the dataset as it is being processed since *record_number* is negative. The input uses the APSHELL conventions as specified in the chapter "APSHELL, UNICOS Scripts for Batch Computing" on page 2.

```
//TORTFLUX JOB CLASS=C,REGION=300K,TIME=2,MSGCLASS=W
// EXEC CRAY
//SYSUT1 DD *
# user=
# QSUB -r TRTFLUX
# QSUB -q day
# QSUB -lT 15
# QSUB -lM 1000Kw
# QSUB -eo
# QSUB
#
set -vxS
APOUTPUT=output
APDEST=rads12
APBANNER=trtflux
APLINES=10000
. APSHELL
STAGEIN IBMBINARY STACK C116.IBM.TEST.RTFLUX
IFBOMB
cat > input <<EOF
 RTFLUX -1  CRAY IBM
EOF
XFTU
cp RTFLUX DUMMY1
cp RTFLUX STACK
STAGEOUT CRAYBINARY DUMMY1 your.choice.dsname \
   'DISP=(NEW,CATLG),UNIT=TEMP,SPACE=(TRK,(10,3),RLSE)' \
   'DCB'
cat > input <<EOF
 RTFLUX  1  BCD CRAY
EOF
XFTU
STAGEOUT CARDS RTFLUX your.choice.dsname \
   'DISP=(NEW,CATLG),UNIT=TEMP,SPACE=(TRK,(10,3),RLSE)' \
   'DCB'
/*
```

Figure 3. Typical Cray FTU Job Converting IBM to Cray Format

Users should remember that if more than one file is to be accessed using `STAGEIN`, a `rm STACK` command must first be specified before subsequent `STAGEIN`s since `STAGEIN` does nothing if the cray file already exists.

Figure 4 shows a typical Cray execution of FTU in which a VAX format ISOTXS dataset [3] is converted to a Cray format version. The format for fetching the VAX binary file differs noteably from that required for fetching IBM binary files.

```
//VAXCRAY JOB CLASS=C,REGION=300K,TIME=2,MSGCLASS=W
// EXEC CRAY
//SYSUT1 DD *
# user=
# QSUB -r VAXCRAY
# QSUB -q day
# QSUB -lT 15
# QSUB -lM 1000Kw
# QSUB -eo
# QSUB
#
set -vxS
APOUTPUT=output
APDEST=rads12
APBANNER=vaxcray
APLINES=10000
. APSHELL
fetch STACK -mVG -fTB \
   -t'anlvg"BXXXXX password"::CC116:[BXXXXX]YOUR_CHOICE.DAT:1'
IFBOMB
cat >> output <<EOF
 ISOTXS 1  CRAY VAX
EOF
XFTU
STAGEOUT CRAYBINARY ISOTXS your.choice.dsname \
   'DISP=(NEW,CATLG),UNIT=TEMP,SPACE=(TRK,(10,3),RLSE)' \
   'DCB'
/*
```

Figure 4. Typical Cray FTU Job Converting VAX to Cray Format

3.2.4 IBM Execution of FTU

A new catalogued procedure APPROC is also now available in the system library `$SYS1.USERPROC.AP` which is used for executing FTU on the IBM computers. Users should include `PROC = AP` on the `//*MAIN` card to access this catalogued procedure. APPROC follows the Fortran 77 convention of using a file name rather than `FTXXF00N` for the `DDNAME`. A typical IBM FTU job which reads a STACK file and prepares binary PWDINT and ZNATDN datasets, and a BCD

equivalent of the RTFLUX dataset is shown below in Figure 5 as an example of using the new procedure. Each of the datasets being processed is documented in Ref. [3]. The PWDINT, RTFLUX, and ZNATDN datasets are located respectively at record numbers 227, 118 and 13 on the STACK file. The ZNATDN dataset is to be edited as it is being copied from the STACK file since the *record_number* is negative.

The APPROC catalogued procedure is listed in Appendix A. Note that when overwriting DD cards, the supplied cards must be in the order in which they appear in the PROC, just as is required when using catalogued procedures which have numbered DD cards. The `PATH` symbolic parameter has been defaulted to `STP021` corresponding to the DIF3D code [2]. In Figure 5, `PATH` was set to `FTU` and the library from which the FTU code was obtained was specified using the `PRELIB` symbolic parameter.

```
//TORTFLUX JOB CLASS=W,REGION=1500K,TIME=2
//*MAIN LINES=50,PROC=AP
//*FORMAT PR,DDNAME=,DEST=RADS12
//*FORMAT PR,DDNAME=SYSUDUMP,DEST=PR0,FLASH=LINE
//*FORMAT PR,DDNAME=FT10F001,DEST=ANLVM.FICHE
// EXEC APPROC,PATH=FTU,
// PRELIB='C116.B05317.REBUS3.TEST.MODLIB'
//FT10F001 DD DUMMY
//PWDINT DD DSN=your.choice.dsname,DISP=(NEW,CATLG),UNIT=TEMP,
//  VOL=SER=TEM401,SPACE=(TRK,(10,3),RLSE),
//  DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//* BINARY DATASET PWDINT
//RTFLUX DD DSN=your.choice.dsname,DISP=(NEW,CATLG),UNIT=TEMP,
//  VOL=SER=TEM401,SPACE=(TRK,(10,3),RLSE),
//  DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//* BCD DATASET RTFLUX
//STACK DD DSN=existing.STACK.dataset,DISP=SHR
//* STACK DATASET CONTAINING FILES PWDINT, RTFLUX, AND ZNATDN
//ZNATDN DD DSN=your.choice.dsname,DISP=(NEW,CATLG),UNIT=TEMP,
//  VOL=SER=TEM401,SPACE=(TRK,(10,3),RLSE),
//  DCB=(RECFM=VBS,LRECL=X,BLKSIZE=6136)
//* BINARY DATASET ZNATDN
//SYSIN DD *
    PWDINT   227  IBM  IBM
    RTFLUX   118  BCD  IBM
    ZNATDN   -13  IBM  IBM
/*
```

Figure 5. Typical IBM FTU Job

3.2.5 VAX Execution of FTU

Figure 6 shows a typical VAX COMMAND file for FTU. If the input data in file `INPUT.DAT;1` contains

```
ISOTXS 1 BCD VAX,
```

the binary ISOTXS dataset [3] in file 1 TEST_ISOTXS.DAT;1 would be converted to BCD form. The COMMAND file in Figure 6 presumes that the FTU code is available in an FTU.EXE file.

```
$SET VERIFY
$SET DEFAULT [BXXXXX]
$ASSIGN OUTPUT.LIS SYS$OUTPUT
$ON ERROR THEN GOTO BOMB1
$TYPE INPUT.DAT;1
$ASSIGN INPUT.DAT;1 SYS$INPUT
$COPY TEST_ISOTXS.DAT;1 STACK.DAT;1
$RUN FTU
$DELETE STACK.DAT;1
$DIR
$CHARGES
$DEASSIGN SYS$OUTPUT
$ON ERROR THEN GOTO BOMB2
$DELETE OUTPUT.LIS;*
$DELETE STACK.DAT;1
$EXIT
$BOMB1:
$ON ERROR THEN GOTO BOMB2
$CHARGES
$DEASSIGN SYS$OUTPUT
$CONVERT/APPEND OUTPUT.LIS FTU.LIS
$BOMB2:
$DELETE OUTPUT.LIS;*
$DELETE STACK.DAT;1
```

Figure 6. Typical VAX FTU COMMAND File

3.2.6 FTU Edits

Figure 7 shows an example of the edit generated by FTU for a trivial NDXSRF dataset [3]. When a record consists of several different arrays, each array is displayed separately starting on a new line along with the record identifier heading such as RECORD 1D, RECORD 2D, etc. The file identification record is referenced as RECORD ID.

```
                                                ====== EDIT OF FILE NDXSRF

                                                                RECORD ID
NDXSRF     6/20/88 1436.4
                                                                RECORD ID
     1
                                                                RECORD 1D
     6       1       6       6       6       0
                                                                RECORD 2D
I1         I2         I3         I4         I5         I6
                                                                RECORD 2D
I1         I2         I3         I4         I5         I6
                                                                RECORD 2D
 0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
                                                                RECORD 2D
 1.0000E+02  1.0000E+02  1.0000E+02  1.0000E+02  1.0000E+02  1.0000E+02
                                                                RECORD 2D
     0       0       0       0       0       0
                                                                RECORD 2D
     6       0       0       0
                                                                RECORD 2D
     1       2       3       4       5       6
                                                                RECORD 2D
     1       2       3       4       5       6
                                                                RECORD 3D
 1.4304E+03  1.5209E+03  1.9555E+03  1.0000E-06  2.1728E+02  1.0864E+02
                                                                RECORD 3D
 1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00
                                                                RECORD 3D
     1       1       1       1       1       1
```

Figure 7. FTU Generated Edit of Dataset NDXSRF

3.2.7 FTU BCD Conversion

Figure 8 shows an example of the BCD equivalent form for the NDXSRF dataset shown in Figure 7. As is the case for the dataset edits, when a record consists of several different arrays, each array is displayed separately starting on a new line along with the record identifier such as `1D`, `2D`, etc. The file identification record is referenced as `ID`. Obviously, if the BCD equivalent for a dataset is generated,

the user may simply list that file as an edit of the dataset rather than having FTU generate an edit of the dataset.

```
ID  NDXSRF     6/20/88 1436.4
ID        1
1D        6        1        6        6        6        0
2D  I1          I2          I3          I4          I5          I6
2D  I1          I2          I3          I4          I5          I6
2D  0.00000000E+00 0.00000000E+00 0.00000000E+00 0.00000000E+00 0.00000
    0.00000000E+00
2D  0.10000000E+03 0.10000000E+03 0.10000000E+03 0.10000000E+03 0.10000
    0.10000000E+03
2D        0        0        0        0        0        0
2D        6        0        0        0
2D        1        2        3        4        5        6
2D        1        2        3        4        5        6
3D  0.14304124E+04 0.15209446E+04 0.19554993E+04 0.10000003E-05 0.21727
    0.10863995E+03
3D  0.10000000E+01 0.10000000E+01 0.10000000E+01 0.10000000E+01 0.10000
    0.10000000E+01
3D        1        1        1        1        1        1
```

Figure 8. BCD Equivalent for Dataset NDXSRF

3.3 Programming Information

FTU has been coded using Fortran 77 conventions and is organized into subroutines, each of which is named for the file which is being processed together with a number of general utility subroutines. Table 2 lists the FTU subroutines with a brief description of the function of each.

The various Cray utility routines utilized by FTU for data format conversion are listed in Table 3 along with a brief description of the function of each. The detailed description of the calling sequence for each as well as implementation information is given in Ref. [7]. The utilities with names beginning with `US` are concerned with IBM format conversion, those beginning with `VX` with VAX format conversion, and the `FP` and `INT` routines with CDC format conversion.

The programming conventions used in writing FTU make extensive use of special "keyword" comment cards surrounding machine-dependent Fortran cards. The coding between a keyword pair is selectively activated or deactivated by a simple preprocessing Fortran program which places a blank or the letter C in column 1 of the bracketed card images [10]. As a simple example, the following code fragment is appropriate for short word machines, and could be converted for long word machines by deactivating the `CSW` and activating the `CLW` keywords.

Table 2. FTU Subroutines

Subprogram	Function
MAIN	Main driver for FTU
SELECT	Reads the user input specifications, positions file STACK, and reads, converts, and writes the file identification record for the dataset being processed
FILEID	Converts the file identification record and writes this record for the dataset being processed
AAFLUX	Processes the AAFLUX dataset
ADJANG	Processes the ADJANG dataset
ANGSRC	Processes the ANGSRC dataset
ATFLUX	Processes the ATFLUX dataset
COMPXS	Processes the COMPXS dataset
DIRANG	Processes the DIRANG dataset
DISFAC	Processes the DISFAC dataset
DLAYXS	Processes the DLAYXS dataset
D3EDIT	Processes the D3EDIT dataset
FIXSRC	Processes the FIXSRC dataset
GEODST	Processes the GEODST dataset
ISOTXS	Processes the ISOTXS dataset
LABELS	Processes the LABELS dataset
NAFLUX	Processes the NAFLUX dataset
NDXSRF	Processes the NDXSRF dataset
NHFLUX	Processes the NHFLUX dataset
PMATRX	Processes the PMATRX dataset
PWDINT	Processes the PWDINT dataset
RAFLUX	Processes the RAFLUX dataset
RTFLUX	Processes the RTFLUX dataset
RZFLUX	Processes the RZFLUX dataset
SFEDIT	Processes the SFEDIT dataset
ZNATDN	Processes the ZNATDN dataset
CNVRTC	Converts character data to character data for the specified format
CNVRTI	Converts integer numbers to integer numbers for the specified format
CNVRTR	Converts floating point numbers to long or short word floating point data for the specified format
BCDC	Converts character binary records into BCD equivalent data
BCDI	Converts integer binary records into BCD equivalent data
BCDR1	Converts single precision binary records to BCD equivalent data
BCDR2	Converts double precision binary records to BCD equivalent data
COPYFI	Reads, converts, and writes records which are of uniform type: all integer, all floating point, or all character data
PRECIS	Checks datasets on short word length machines to determine word length
VSRIN	Reads VAX segmented binary records
VSROUT	Writes VAX segmented binary records

Table 3. Cray Format Conversion Utility Routines used by FTU

USCCTC	Converts IBM EBCDIC data to ASCII data
USCCTI	Converts ASCII data to IBM EBCDIC data
USDCTC	Converts IBM 64-bit double-precision floating-point numbers to Cray single-precision numbers
USICTC	Converts IBM INTEGER*2 and INTEGER*4 numbers to Cray 64-bit integer numbers
USSCTC	Converts IBM 32-bit single-precision floating-point numbers to Cray 64-bit single-precision numbers
USSCTI	Converts Cray 64-bit single-precision floating-point numbers to IBM 32-bit single-precision numbers
VXDCTC	Converts VAX 64-bit D format numbers to Cray single-precision numbers
VXDCTI	Converts Cray 64-bit single-precision floating-point numbers to VAX D format double-precision numbers
VXSCTC	Converts VAX 32-bit single-precision floating-point numbers to Cray 64-bit single-precision numbers
VXSCTI	Converts Cray 64-bit single-precision floating-point numbers to VAX F format single-precision numbers
VXICTC	Converts VAX INTEGER*2 or INTEGER*4 numbers to Cray 64-bit integers
VXICTI	Converts Cray 64-bit integers to VAX INTEGER*2 or INTEGER*4 numbers
FP6064	Converts CDC 60-bit single-precision floating-point numbers to Cray 64-bit single-precision numbers
FP6460	Converts Cray 64-bit single-precision floating-point numbers to CDC 60-bit single-precision numbers
INT6064	Converts CDC 60-bit integers to Cray 64-bit integers
INT6460	Converts Cray 64-bit integers to CDC 60-bit integers

```
CSW
      DOUBLE PRECISION A,B
CSW
CLW
C     REAL A,B
CLW
```

Note that further comments concerning the use of keyword comment cards can be found in the next chapter.

Due to the word length differences between the Cray and the IBM or VAX computers as well as the special nature of VAX segmented records [9], special I/O routines are required by FTU to accomplish the data transfer. The Cray utilities READP, WRITEP, and WRITE [7] are used to read files which were written on the IBM or VAX computers, and to write files which are to be read on the VAX computer. READP and WRITEP provide the ability to read and write partial records, and WRITE is used to write end-of-record marks for VAX format files. Subroutines VSROUT and VSRIN [8] are used to add and

remove, respectively the extra segment control bytes required VAX segmented records [9]. IBM format files are written using ordinary Fortran WRITE statements.

3.4 FTU Limitations

FTU is designed to be portable to Cray, IBM, VAX, and CDC computers and has been implemented on the Cray X-MP under Unicos, IBM 3033 under MVS, and VAX 8700 under VMS.

When running on the Cray computer under Unicos, FTU permits binary data to be converted to the format required by IBM, VAX, and CDC computers, and binary data formats which were written on these computers to be converted to Cray format conventions. FTU has been tested for binary file format conversion between the Cray and IBM systems for all of the datasets specified in Table 1 on page 17. Conversion of only the binary ISOTXS dataset format has been tested between the Cray and VAX systems.

When running on computers other than the Cray, FTU does not provide binary file format conversion but does provide the option of generating a BCD equivalent form for the various datasets.

FTU provides the ability to copy binary files without format conversion from a file on which a number of datasets have been "stacked" on any of the computers.

FTU has not as yet been implemented on a CDC system nor has testing of binary format conversion between the Cray and CDC systems been undertaken.

Chapter 4

A MODULAR PROGRAMMING SYSTEM FOR ENGINEERING PHYSICS DIVISION CODES

4.1 Introduction

Current Engineering Physics production codes exist in a modular environment on the IBM system [10]. In such an environment major computational application programs are constructed using a collection of modules coupled together via a driver module (path-driver) and interface files. The path-driver serially invokes other modules via a call to LINK (an IBM assembler routine). LINK permits argument passing and an arbitrary, but nonrecursive, nesting of modules. The environment also permits special utility modules (marked REUSable) to be permanently resident in memory. These special modules may be LINKed by all other modules and provide useful data communication and synchronization functions.

The flexibility and economic advantage experienced with the current Engineering Physics modular system on the IBM 3033 motivates the development of a Fortran 77 based modular system whose architecture is as machine independent as possible. This modular system uses a path-driver module to manage the serial execution of application modules. Flexibility arises from module independence. Changes to a single module are immediately available to all path-drivers invoking that module. The ability to preempt production module libraries with alternate module libraries is a convenient and economic tool that provides specialized applications and simplifies code maintainence and development. This flexibility increases file space requirements and, on systems that require file staging, staging costs. Modular system storage overhead arises from the system routines loaded with each module regardless of its size. Storage overhead for a comparable collection of standalone codes arises from the common set of building block modules loaded with each code.

The remainder of this chapter describes a Fortran 77 modular system implemented on a CRAY X-MP (UNICOS, CFT77), an IBM 3033 (MVS, FORTVS) and a VAX 8700 (VMS, FORT77).

4.2 Functional Requirements for a Fortran 77 based Modular System

The following functional requirements are needed by a modular system:

1. At least one level of Fortran modules must be callable from a Fortran path-driver module.

2. Communication between modules must be via interface files; argument lists are not allowed.

3. Output from all modules should be synchronized (i.e. appear in chronological order).

4. Debugging capabilities comparable to those available for standalone codes are desirable.

5. An implementation that avoids or minimizes custom assembler code and features available on only one machine is desirable for code portability.

Requirements (1) and (2) minimize the machine-dependent features needed to implement a modular system. Requirement (3) is obvious and requirement (4) is expected in any robust implementation. They are mentioned to emphasize the fact that machine-dependent coding may be required to meet these goals. Though not essential for modular systems, the portability requirement (5) has proved highly successful for exporting Engineering Physics Division codes to different computers.

4.3 Implementation Overview

The modular system is comprised of Fortran 77 programs which fall into one of two functional categories, path-driver modules and application modules. A path-driver module controls the order (path) in which modules are called. Some drivers (e.g. REBUS-3) contain elaborate logic, others (e.g. DIF3D) contain little or no path-dependent logic. A utility subroutine LINKMD is called by the path-driver to link to application modules.

All data communication between modules must be via interface files to improve code portability. Application codes typically require several interface files. The file name MODCOM together with logical unit number 1 is reserved for a special "system" interface file that passes essential "system" data (e.g current page number) between modules. The utility subroutine MSYNCW and its entry MSYNCR synchronize the path driver and application modules by respectively writing and reading "system" data on MODCOM whenever execution control changes hands.

The two utility routines LINKMD and MSYNCW isolate the machine-dependent aspects of the modular system. LINKMD which must be called only by a path-driver module performs several functions. It links a module (and invokes debug commands if needed), it synchronizes "system" data, and it ensures the continuity of printed output. Two module names, M_INIT and M_END, are reserved for control sentinels and do not correspond to actual modules. The path-driver must initialize the modular system by calling LINKMD with module name M_INIT. Upon completion of the path-driver a final LINKMD call with module name M_END is required to wrapup the modular system.

System synchronization calls on behalf of the path-driver module are made from subroutine LINKMD. It calls MSYNCW prior to linking a module, and it calls MSYNCR just after the application module exits. Upon entry, each applications module must immediately call MSYNCR to complete the system synchronization. A call to MSYNCW just before exit from an application module reinstates the current system synchronization data on MODCOM. MSYNCW and MSYNCR require a single sentinel argument to indicate the calling module's identity (i.e. 0 indicates a path-driver, 1 indicates an application module).

Chronologically ordered printed output from all modules is ensured by subroutines MSYNCW, LINKMD and FINOUT. Two printed output files (units 6 and 10) are supported to accomodate Engineering Physics Division codes that normally use two output streams. The default connections for units 6 and 10 must be overridden in some systems (e.g. UNICOS and VMS) to achieve output continuity. The third utility subroutine FINOUT performs machine-dependent tasks (if needed) that flush the path-driver output files just prior to path-driver termination. FINOUT is always called from LINKMD and may also be called from an error handling routine just prior to a program controlled abnormal exit. FINOUT uses the common block named MODFLG.

Examples of a path-driver and an applications module illustrating this modular system implementation are in Figure 9 and in Figure 10. Details of the implementation are explained in the sections that follow.

```
CDECK MODDRV
CF77
      PROGRAM MODDRV
CF77
CF77-SW
C     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CF77-SW
CF66-SW
C     IMPLICIT REAL*8 (A-H,O-Z)
CF66-SW
      COMMON /IOPUT/ NIN, NOUT, NOUT2
C
C     SET PRINTED OUTPUT FILE UNIT NUMBERS
C
      CALL LINKMD ( 'M_INIT  ' )
C
C     END PATH DRIVER INITIALIZATION
C
C     ARGONNE EP CODES NORMALLY EMPLOY SCAN AND STUFF MODULES
C
      CALL LINKMD ( 'SCAN    ')
      CALL LINKMD ( 'STUFF   ')
C
C     NOW CALL APPLICATION MODULES (PATH LOGIC MAY APPEAR HERE ALSO).
C
      CALL LINKMD ( 'MOD1    ')
      CALL LINKMD ( 'MOD2    ')
C
C     PATH DRIVER WRAPUP (Last statement in path driver)
C
      CALL LINKMD ( 'M_END   ')
      STOP
      END
```

Figure 9. Simple Path-Driver Module in a Modular System

4.4 Synchronization of Printed Output

Subroutine MSYNCR and entry MSYNCW must be called at the beginning and end of each applications module (including "dummy" UDOIT modules), respectively, to obtain chronologically ordered printed output from path driver modules and applications modules.

```
CDECK MOD1
CRAY-MOD
      PROGRAM MOD1
CRAY-MOD
CVMS-MOD
C     PROGRAM MOD1
CVMS-MOD
CIBM-MOD
C     SUBROUTINE MOD1
CIBM-MOD
CSA
C     SUBROUTINE MOD1
CSA
CF77-SW
C     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CF77-SW
CF66-SW
C     IMPLICIT REAL*8 (A-H,O-Z)
CF66-SW
      COMMON /IOPUT/ NIN, NOUT, NOUT2
      CALL MSYNCR ( 1 )
C
C     MODULE APPLICATION CODING BEGINS HERE
C
C     . . .
C
C
C     APPLICATION MODULE CODE ENDS HERE
C
      CALL MSYNCW ( 1 )
CSA
C     RETURN
CSA
CIBM-MOD
C     RETURN
CIBM-MOD
      END
```

Figure 10. Simple Applications Module in a Modular System

4.4.1 Synchronization of Printed Output In the UNICOS System

All printed output in the UNICOS implementation ultimately appears on two files, `$APOUTPUT` (see Chapter 2) and `fort.10`. As each module is linked its stdout (unit 6) is **redirected** and **concatenated** to `$APOUTPUT`; unit 10 is connected to `fort.10`. After an applications module exits LINKMD concatenates the current contents of `fort.10` to `fort10` where it remains until FINOUT ultimately renames `fort10` to fort.10 just prior to the final exit from the path driver module.

The shell variable `APOUTPUT` in LINKMD is consistent with the APSHELL variable of the same name. If APOUTPUT is not defined, the modular system uses a default name of `output`. The APSHELL environment is not required to run the modular system.

In order to synchronize the printed output from the path-driver and application modules, LINKMD connects (via an OPEN statement) two temporary file names `path6` and `path10` to units 6 and 10 in the path driver. An ENDFILE statement deletes the contents of each file. The path driver must not attempt to connect unit 6 to `$APOUTPUT` if chronologically ordered output is to be maintained. Prior to invoking an application module, LINKMD disconnects `path6` and `path10`. Upon entry to an application module, a call to MSYNCR connects unit 10 to `fort.10` and clears `fort.10`. Unit 6 is already connected to stdout by the module linking logic. Then MSYNCR copies any path driver output on `path6` and `path10` to units 6 (stdout) and 10 (`fort.10`), respectively. Unit 1 is used for temporary connections to `path6` and `path10`. After MSYNCR exits the application module continues printing on units 6 and 10. When the application module exits, LINKMD concatenates the current contents of `fort.10` to `fort10`. Then `path6` and `path10` are reconnected to units 6 and 10, respectively, and cleared via an ENDFILE statement. Output created after the last application module has exited is copied to the corresponding output files by subroutine FINOUT prior to final exit from the path driver. FINOUT is also called by subroutine ERROR prior to an abnormal termination. In all cases FINOUT ultimately renames `fort10` to `fort.10` just prior to exit.

4.4.2 Synchronization of Printed Output in the VAX/VMS System

The printed output in the VAX/VMS system is synchronized by overriding the default connections to units 6 and 10 (FOR006 and FOR010, respectively) with OPEN statements that include the specification ACCESS=APPEND. Appropriate OPEN and CLOSE statements in LINKMD and MSYNCR before and after module entrance and exit cause output to be concatenated on the same file in chronological order.

4.4.3 Synchronization of Printed Output on the IBM MVS System

Output synchronization occurs using the default connections FT06F001 and FT10F001 for units 6 and 10, respectively. One side effect that is not yet understood occurs when a symbolic dump is triggered from a called module. The normal symbolic dump appears to be followed by a second symbolic dump that repeats the subset of routines that are part of the path driver module.

4.5 Module Linking

Fortran 77 has no provision for linking independent program modules. Consequently this machine-dependent function is isolated in LINKMD, a Fortran 77 subroutine with one CHARACTER*8 argument. A path-driver serially links a module by calling LINKMD with the desired upper case module name as the argument. Module names M_INIT and M_END are reserved for control purposes. The first call to LINKMD must use the name M_INIT to initialize the modular system. During this initialization call variables NOUT and NOUT2 in common block IOPUT are set to unit numbers 6 and 10, respectively. Calling LINKMD with module name M_END signals that an exit from the path-driver is imminent and triggers modular system termination tasks. On systems where symbolic dumps are not automatically generated, LINKMD is also used to isolate machine-dependent invocations of symbolic debugging commands that are triggered when an application module terminates abnormally.

4.5.1 Module Linking in the UNICOS System

By convention module names in UNICOS must be lower case. Consequently, LINKMD calls subroutine SHFTLO to convert the module name in its argument from upper case to lower case and store it in the CHARACTER*8 variable named MODULE. Modules are linked by calling the system function ISHELL [7]. A single argument of type character is used to pass a command string to a UNICOS bourne shell (a "child" of the shell in which the current module is executing). The current process (i.e. the path driver module) waits until the shell has completed executing the command string and receives the exit status in the function value returned by ISHELL (supplying an & after the command sent to ISHELL permits asynchronous module execution, but this usage is not supported in the current modular implementation). Calls to ISHELL from LINKMD and FINOUT include combinations of the following UNICOS command string examples below and may also include the Fortran 77 operator '// ' which concatenates character variables and character string literals.

- `$APTRACE`

  Optional user supplied shell commands may be put into shell variable `APTRACE` for diagnostic purposes (e.g. see Figure 13). Commands in `APTRACE` are executed prior to every module invocation if APTRACE has been exported (export typically done by applications scripts).

- `MODULE // '>> ${APOUTPUT:-output}'`

  The `MODULE` command invokes the desired module. The current implementation redirects and concatenates stdout from `MODULE` to `$APOUTPUT` (default is `output`). The shell variables `APTRACE` and `APOUTPUT` are available to the shell spawned by ISHELL only if they have been exported prior to executing the path driver module. The applications scripts like XDIF3D include the appropriate export commands.

- `'cat fort.10 >> fort10 '`

  Upon exit from `MODULE`, the printed output for logical unit 10 (default filename `fort.10`) is concatenated to `fort10` from `fort.10`.

- ```
  'debug -B -d 20,5,5 -s `hash '  // MODULE // ' ; hash | grep ' //
  MODULE // ' | cut -f3` >> sysudump ; '
  ```

  The debug command concatenates symbolic dump output onto a file named `"sysudump"` using the file named `"core"` that is generated when a module terminates abnormally. The `hash` commands are used to determine the search path directory where `MODULE` is found. LINKMD invokes debug whenever ISHELL returns a nonzero return code following an application module invocation.

- `'cat path6 >> ${APOUTPUT:-output} ; cat path10 >> fort10 '`

  Just prior to the final exit from a path driver append `path6` to `$APOUTPUT` and append `path10` to `fort10`.

- `'rm path6 path10 ; mv fort10 fort.10 '`

  Just prior to the final exit from FINOUT remove auxiliary output files `path6` and `path10`, then rename `fort10` to `fort.10`.
  ```

4.5.2 Module Linking in the VAX/VMS System

Module linking is accomplished by writing a VMS DCL command file to invoke the desired module then spawning a subtask that executes the command file. The DCL text required to execute a module is `$RUN MODULE` where `MODULE` denotes the name of the module to be executed. LINKMD creates the appropriate command file text and passes it to subroutine VSPAWN which writes the command file (PATH_SPAWN.DAT) and spawns a subtask by calling the LIB$SPAWN run-time system function. A logical unit number for the command file is obtained by system function LIB$GET and later released by LIB$FREE.

4.5.3 Module Linking in the IBM MVS System

Load modules in the IBM MVS system are linked from LINKMD by calling assembler routine LINK [1]. The 8 character module name is passed as the only argument to LINK. Although LINK permits subroutine arguments in the call, the modular system machinery does not permit or use it.

4.6 Local Code Conversion Considerations for the New Modular System

The rudiments of the modular system are illustrated by the simple path-driver module listed in Figure 9 on page 29 and the simple applications module listed in Figure 10 on page 30.

4.6.1 CONVTCD Keyword Changes

Several changes to our current coding practices [10] are required by the new modular implementation. The REUSable modules SYS001 thru SYS005 have been eliminated. The keyword table for the CONVTCD utility now includes a new keyword CMOD to designate a modular implementation. CMOD and CSA are mutually exclusive keywords. As a consequence the past practice of using CANL to denote IBM modular must now be changed to CMOD-F66. CANL must now be qualified with CIBM, CUNC or CVMS when code modifications only apply to a particular Argonne computer.

The keyword sentinels now required by CONVTCD for the four possible modular systems are:

1. *IBM66/MVS*

   CSW C1LV CENT CIBM CANL CF66 CDYN CMOD

2. *IBM77/MVS*

   CSW C1LV CENT CIBM CANL CF77 CDYN CMOD

3. *CRAY/UNICOS*

   CLW C1LV CENT CRAY CANL CF77 CDYN CMOD CVEC CSEG CUNC

4. *VAX/VMS*

   CSW C1LV CENT CVMS CANL CF77 CDYN CMOD

4.6.2 Path-Driver Changes

All Fortran 77 path-driver modules should include PROGRAM statements for unique identification in an object code library. Module linking calls must be made via a call to subroutine LINKMD which has one argument, a CHARACTER*8 variable containing an upper case module name. As illustrated in Figure 9 on page 29 each path-driver initializes the modular system by calling LINKMD (with argument 'M_INIT'). The initialization sets NOUT and NOUT2 to 6 and 10, respectively. NOUT is the logical unit number for the primary printed output stream, NOUT2 is the logical unit number for the secondary output stream, and NIN is the logical unit number for the standard input file. A call to LINKMD (with argument 'M_END') is required just before the final exit from the path-driver. M_INIT and M_END are reserved words that are used as sentinels to control modular system initialization and wrapup. The modules SCAN and STUFF must now be linked from the path driver module. Prior practice required direct calls to SCAN and STUFF dummy drivers that linked to modules SYS001 or SYS002.

4.6.3 Applications Module Changes

A keyword change is also required in the driving subprogram of each applications module. Depending on the system and the type of implementation (modular or standalone) the driving subprogram of a module must begin with either a SUBROUTINE statement or a PROGRAM statement. A corresponding appropriate termination statement (RETURN or the absence of a RETURN) is also required in the driving subprogram. As illustrated in Figure 10 on page 30, UNICOS and VAX VMS modular systems require a PROGRAM statement. All other implementations currently require a SUBROUTINE statement.

Subroutine calls to MSYNCR and MSYNCW must also appear in the driver of every applications module including dummy UDOIT modules. MSYNCR should be the first executable statement and MSYNCW should be the last statement executed just before exit from the module. Because MSYNCR initializes TIMER and the common block variables (NOUT=6 and NOUT2=10) in /IOPUT/ and (KOUT=6 and KOUT2=10) in /PTITLE/, applications modules no longer need perform these initializations. The variable NIN is initialized from the path driver and passed to applications modules via the MODCOM system interface file. Output is hardwired to units 6 and 10 in MSYNCR to reduce the potential for printed output loss in the event of an abnormal termination. Applications modules can still temporarily disable output to units 6 or 10 by setting the appropriate local common block variables to 0 after the call to MSYNCR.

4.6.4 Changes to Utility Subroutines in the Existing Libraries

- *ARCBCD*

  Member ARCBCD contains source for both the SCAN and STUFF modules and was changed to include appropriate PROGRAM, SUBROUTINE and RETURN statements in the SCAN and STUFF routines. Calls to subroutines MSYNCW and MSYNCR were also added. The modules SYS001 and SYS002 were eliminated. Shared data retained in memory by SYS001 is now obtained by MSYNCR from MODCOM.

- *DOPC, DRED, LUNREF AND MAKDDN*

  Logical unit number 1 has been added to the list of reserved special purpose unit numbers. It is used by LINKMD and MSYNCR for temporary connection to MODCOM while reading and writing "system" synchronization data and for temporary connection to `path6` and `path10` while synchronizing printed output on the UNICOS implementation. CONVTCD keywords

CF77-SA were changed to CF77 in order to permit dynamic logical unit number allocation with modular and standalone implementations. Special coding bracketted by CF77-ANL was inactivated by comment cards since the CANL keyword no longer denotes the IBM modular implementation. Two hybrid code systems TWODANT and DPT are affected by this change.

- *ERROR*

  A call to the new utility routine FINOUT was added to ERROR to empty and delete the path driver output files in the event a fatal error exit is detected by subroutine ERROR when it is called from the path driver.

- *LINES AND PGCNTR*

  Logic was added to PGCNTR to permit reinitialization of the common block CPGCNT used by LINES and PGCNTR. The reinitialization is done via calls to PGCNTR (with IENTRY=3) from MSYNCR whenever a module is entered or a path-driver is reentered. Common block CPGCNT and the argument list to PGCNTR were revised as a consequence. Reusable module SYS004 is eliminated. Shared data for CPGCNT is now obtained by MSYNCR from MODCOM.

- *SEEK*

  Logic associated with the SEEK initialization (SEEK option 3) was revised to permit SEEK reinitialization whenever a module was entered or the path-driver reentered upon exit from a module. Reinitialization is triggered by a SEEK option 3 call whenever it is accompanied by a dataset name of RESEEK in argument 1. Reusable module SYS003 is eliminated. Shared data formerly held by SYS003 is obtained by MSYNCR from MODCOM.

- *TIMER*

  A number of timing functions previously surrounded by CANL CONVTCD keywords were changed to CANL-IBM to permit the more general usage of CANL. A subroutine SECOND for IBM F77 modular systems was added to the TIMER librarian module. The reusable module SYS005 is eliminated for IBM F77 modular implementations. The new subroutine SECOND uses the TLEFT assembler routine to compute elapsed time. An entry TSYNC was added to TIMER to synchronize timing data when separate clocks are used in the path-driver and applications modules.

4.7 Modular Library Maintenance and Applications

Summarized in this section will be typical examples of how modular libraries are maintained for the systems currently supported.

4.7.1 Modular Library Maintenance on the UNICOS System

The Cray UNICOS modular production libraries stored on the IBM disks are:

- C116.CRAY.SYSLIB
- C116.CRAY.SEGLIB
- C116.CRAY.OVERLAY

C116.CRAY.SYSLIB and C116.CRAY.SEGLIB are object libraries managed via the UNICOS `bld` command. Executable load modules are created from these object libraries using the UNICOS `segldr` command. Overlayed modules require a `segldr` directives file to specify the overlay tree structure. Each directives file is stored in C116.CRAY.OVERLAY, a PDS library file on the IBM disks with a member name ending with TR (e.g. GNIP4CTR) (see Figure 11).

The UNICOS command `ar` is used to manage libraries of UNICOS load modules when they are stored on the IBM disks, in a manner similar to the way the UNICOS `bld` command is used to manage object code libraries. The heavily used Engineering Physics production modules reside permanently on the Cray as files in the directory /n1/b05432/cccc_modlib. The directory is included in the search path by APSTART (see Chapter 2) so that it is available to all module execution scripts when using APSHELL. Figure 12 is a script fragment that compiles and links a typical load module (e.g. GNIP4C) and replaces the load module in a module library.

```
TREE
D4(D4C1,D4C2,D4C3,D4C4,D4C5)
ENDTREE
SEGMENT=D4
MODULES=HMG4C
SEGMENT=D4C1
MODULES=OVL1
SEGMENT=D4C2
MODULES=OVL2
SEGMENT=D4C3
MODULES=OVL3
SEGMENT=D4C4
MODULES=OVL4
SEGMENT=D4C5
MODULES=OVL5
ENDSEG
```

Figure 11. Typical segldr Overlay Tree Directives File

4.7.2 Modular Applications

Figure 13 is a UNICOS script fragment illustrating typical production use of load module libraries. The PRELIB and POSTLIB commands may be used to include additional load module libraries from the IBM disks to the search path. The PREPATH and POSTPATH commands may be used to include additional existing UNICOS directories to the search path. As noted in the command descriptions for PRELIB and POSTLIB in Chapter 2, the order in which module libraries and directories are specified indicates their precedence. For example, to override a dummy UDOITn module in C116.CRAY.MODLIB with a UDOITn from an alternate library C116.CRAY.UDOIT.MODLIB, a user must specify

```
PRELIB C116.CRAY.UDOIT.MODLIB
```

Following the completion of the load module script (XDIF3D) the search path remains in its modified state unless restored by appropriate PRELIB, POSTLIB, PREPATH or POSTPATH commands.

```
STAGEIN CARDS source.f    B20245.FORTRAN.SOURCE
STAGEIN CARDS segtree     B20245.CRAY.MODULAR.OVERLAY(GNIP4CTR)
STAGEIN SEGLIB syslib.a   C116.CRAY.MODULAR.SYSLIB
STAGEIN SEGLIB seglib.a   C116.CRAY.MODULAR.SEGLIB

# truncate directives before sequence numbers in cols. 73-80
mv segtree segtree1
cut -c1-72 segtree1 > segtree; rm segtree1

cft77 -e DIxs source.f
IFBOMB echo cft77_errors
cat source.l >> output
rm source.l source.f

bld rv seglib.a source.o    >> output
IFBOMB
bld tv seglib.a  ; rm source.o

STAGEOUT SEGLIB  seglib.a  C116.CRAY.MODULAR.SEGLIB /
 'UNIT=PERM,SPACE=(CYL,(14,1)),DISP=(NEW,CATLG)' 'DCB'
IFBOMB

segldr -o gnip4c -e GNIP4C -M output -i segtree \
 -D "ECHO=ON;CASE=MIXED;MAP=ADDRESS;USX=WARNING;" \
 -l ./seglib.a,./syslib.a,$ANLUTIL \
 /dev/null >> output
IFBOMB
chmod a+rx gnip4c

# replace gnip4c module in the production module directory
cp gnip4c /n1/b05432/cccc_modlib
IFBOMB
ls -l /n1/b05432/cccc_modlib

# replace gnip4c module in a library named modlib;  list its directory
#STAGEIN ARLIB  modlib    C116.CRAY.MODULAR.MODLIB
#ar ru  modlib  gnip4c
#IFBOMB
#ar tv  modlib
#STAGEOUT ARLIB modlib  C116.CRAY.MODULAR.MODLIB
```

Figure 12. UNICOS Script Fragment for Updating a MODLIB

```
# Specify list of ar module libraries located on the IBM disk farm
# to be PRELIBed or POSTLIBed (if needed) into the search path.

PRELIB  C116.CRAY.DEBUG.MODLIB
PRELIB  C116.CRAY.UDOIT.MODLIB
POSTLIB C116.CRAY.GRAPHIC.MODLIB

STAGEIN CARDS input  B20245.JOBX.INPUT
IFBOMB

# Following line is optional. It is useful to trace module execution
# while debugging.  Other commands could also be supplied here.
APTRACE=' set -vxS '

# Do not redirect output for a path driver module.  Printed output is
# ultimately directed to $APOUTPUT (default is output) and fort.10.
# Symbolic dump output appears on the sysudump file.
# Execute the APSHELL script XDIF3D which invokes the DIF3D code.
# XDIF3D exports the shell variables APOUTPUT and APTRACE.
XDIF3D input

# Concatenate symbolic dump trace (if any) and fort.10 to $APOUTPUT
[ -r sysudump ] && cat sysudump >> $APOUTPUT
cat fort.10 >> $APOUTPUT
IFBOMB
```

Figure 13. Sample Production Job Script Fragment for UNICOS Modular System

ACKNOWLEDGEMENTS

The authors would particularly like to thank Jeff Doak of Cray Research for help in understanding the Cray/Unix system, and Alan Hinds for his assistance in implementing the modular system. Many other people from the Computing and Telecommunications Division also contributed their advise and assistance to this work.

REFERENCES

1. L. C. Just, H. Henryson, II, A. S. Kennedy, S. D. Sparck, B. J. Toppel, and P. M. Walker, "The System Aspects and Interface Data Sets of the Argonne Reactor Computation (ARC) System", ANL-7711, Argonne National Laboratory (April 1971).

2. K. L. Derstine, "DIF3D: A Code to Solve One-, Two-, and Three- Dimensional Finite-Difference Diffusion Theory Problems", ANL-82-64, Argonne National Laboratory (April 1984).

3. R. Douglas O'Dell, "Standard Interface Files and Procedures for Reactor Physics Codes, Version IV", UC-32, Los Alamos National Laboratory (September 1977).

4. H. Henryson II, B. J. Toppel, and C. G. Stenberg, " MC^2–2 : A Code to Calculate Fast Neutron Spectra and Multigroup Cross Sections", ANL-8144, Argonne National Laboratory (June 1976).

5. B. J. Toppel, Private Communication.

6. B. J. Toppel, "A Users Guide for the REBUS-3 Fuel Cycle Analysis Capability", ANL-83-2, Argonne National Laboratory (March 1983).

7. "Programmers's Library Reference Manual", SR-0113C, Cray X-MP and Cray-1 Computer Systems, Cray Research,Inc. (June 1987).

8. Jeff Doak, Private Communication.

9. "VAX/VMS User's Manual", Digital Equipment Corporation (April 1986).

10. C. H. Adams, K. L. Derstine, H. Henryson II, R. P. Hosteny, and B. J. Toppel, "The Utility Subroutine Package used by Applied Physics Division Export Codes", ANL-83-3, Argonne National Laboratory (April 1983).

Appendix A

IBM CATALOGUED PROCEDURE APPROC

The APPROC catalogued procedure shown in Figure 14 represents a generic procedure intended for use with any of the EP production codes which have been modified to make use of Fortran 77 conventions. In particular, various symbolic parameters are provided for the various datasets with which the user may be concerned.

```
//APPROC PROC  AACYL=5,AADSP='(,DELETE)',AAFLUX='&AAFLUX',
//             ATCYL=5,ATDSP='(,DELETE)',
//             ATFLUX='&ATFLUX',ATVOL=,DEST='*',DEST2=F,DMPDEST=F,
//             BLKTYP=CYL,CMPDSP='(,DELETE)',COMPXS='&COMPXS',
//             CXSCYL=3,DIFDSP='(,DELETE)',DIF3D='&DIF3D',
//             DLADSP='(,DELETE)',DLAYXS='&DLAYXS',D3DSP='(,DELETE)',
//             D3EDIT='&D3EDIT',FDCCYL=20,FLXCYL=1,
//             GEODSP='(,DELETE)',GEODST='&GEODST',HALFTRK=6136,
//             ISOCYL=1,ISODSP='(MOD,KEEP)',ISOVOL=,
//             ISOTXS='&ISOTXS',LABDSP='(,DELETE)',LABELS='&LABELS',
//             MODEDCB='(RECFM=F,BLKSIZE=23220)',
//             MODLIB1='SYS1.DUMMYLIB',
//             MODLIB2='C116.CCCC.MODLIB',
//             NACYL=5,NADSP='(,DELETE)',NAFLUX='&NAFLUX',NAVOL=,
//             NDXDSP='(,DELETE)',NDXSRF='&NDXSRF',
//             NHCYL=5,NHDSP='(,DELETE)',
//             NHFLUX='&NHFLUX',NHVOL=,PATH='STP021',
//             PMADSP='(,DELETE)',PMATRX='&PMATRX',
//             POSTLIB='SYS1.DUMMYLIB',PRELIB='SYS1.DUMMYLIB',
//             PSICYL=5,PSUCYL=3,PWDDSP='(,DELETE)',QRTRTRK=3064,
//             RACYL=5,RADSP='(,DELETE)',RAFLUX='&RAFLUX',REGN=1000K,
//             RTCYL=5,RTDSP='(,DELETE)',RTFLUX='&RTFLUX',
//             RTVOL=,TIMLIM='(600,0)',RZDSP='(,DELETE)',
//             RZFLUX='&RZFLUX',SFDSP='(,DELETE)',SFEDIT='&SFEDIT',
//             SRFCYL=12,STADSP='(,DELETE)',STACK='&STACK',
//             TWELTRK=1016,UNITS=BATCHDSK,UNITSCR=SASCR,
//             XSISO=NULLFILE,XSISO2=NULLFILE,XSIVOL=,XSI2VOL=,
//             ZNADSP='(,DELETE)',ZONCYL=1
//*
//*    ****************************************************************
//*    *                                                              *
//*    *   CATALOGUED PROCEDURE FOR ENGINEERING PHYSICS DIVISION      *
//*    *   FORTRAN 77 CODES.    ***  11/29/88  ***                    *
//*    *                                                              *
//*    ****************************************************************
//*
//*   CYLINDER ALLOCATIONS ARE FOR 3330 DEVICES
//*   IF OTHER DEVICES ARE USED CHANGE PROC PARAMETER
//*   MODEDCB=(RECFM=F,BLKSIZE=XXXXXX) TO THE APPROPRIATE
//*   BLOCK SIZE (13030 ON 3330, 19069 ON 3350, 23220 ON 3380).
//*   NOTE THAT THE NUMBER OF TRACKS PER CYLINDER ON THESE
//*   DEVICES IS 19 ON 3330, 30 ON 3350, AND 15 ON 3380.
//*  ********************************************************************
//*
//*  PARAMETER    DEFAULT VALUE              USAGE
//*  =========    =============              =====
//*
//*   PATH          STP021           PROGRAM NAME (EXEC)
//*   TIMLIM        (600,0)          STEP TIME LIMIT (EXEC)
```

Figure 14. IBM Catalogued Procedure APPROC

```
//*   REGN          1000K              STEP REGION SIZE (EXEC)
//*   MODLIB1       SYS1.DUMMYLIB      SECOND STEP LIBRARY (STEPLIB)
//*   MODLIB2       C116.CCCC.MODLIB CCCC SYSTEM LIBRARY (STEPLIB)
//*   PRELIB        SYS1.DUMMYLIB      FIRST STEP LIBRARY (STEPLIB)
//*   POSTLIB       SYS1.DUMMYLIB      LAST STEP LIBRARY (STEPLIB)
//*   DEST          *                  OUTPUT DEST. (FT06)
//*   DEST2         F                  OUTPUT DESTINATION (FT10)
//*   DMPDEST       F                  ROUTE DUMP TO FICHE (SYSUDUMP)
//*   AACYL         5                  NO. OF CYL. FOR AAFLUX
//*   ATCYL         5                  NO. OF CYL. FOR ATFLUX
//*   AADSP         (,DELETE)          DISPOSITION OF AAFLUX
//*   ATDSP         (,DELETE)          DISPOSITION OF ATFLUX
//*   ATFLUX        &ATFLUX            DSN FOR DATASET ATFLUX
//*   ATVOL         ------             VOLUME FOR ATFLUX
//*   CMPDSP        (,DELETE)          DISPOSITION OF COMPXS
//*   DIFDSP        (,DELETE)          DISPOSITION OF DIF3D
//*   DLADSP        (,DELETE)          DISPOSITION OF DLAYXS
//*   D3DSP         (,DELETE)          DISPOSITION OF D3EDIT
//*   GEODSP        (,DELETE)          DISPOSITION OF GEODST
//*   ISOCYL        1                  NO. CYL. FOR ISOTXS
//*   ISODSP        (MOD,KEEP)         DISPOSITION OF ISOTXS
//*   ISOTXS        &ISOTXS            DSN FOR DATASET ISOTXS
//*   ISOVOL        ------             VOLUME FOR ISOTXS
//*   LABDSP        (,DELETE)          DISPOSITION OF LABELS
//*   NADSP         (,DELETE)          DISPOSITION OF NAFLUX
//*   NAFLUX        &NAFLUX            DSN FOR NAFLUX
//*   NAVOL         ------             VOLUME FOR NAFLUX
//*   NDXDSP        (,DELETE)          DISPOSITION OF NDXSRF
//*   NHCYL         5                  NO. OF CYL. FOR NHFLUX
//*   NHDSP         (,DELETE)          DISPOSITION OF NHFLUX
//*   NHFLUX        &NHFLUX            DSN FOR NHFLUX
//*   NHVOL         ------             VOLUME FOR NHFLUX
//*   PMADSP        (,DELETE)          DISPOSITION OF PMATRX
//*   PWDDSP        (,DELETE)          DISPOSITION OF PWDINT
//*   RACYL         5                  NO. OF CYL. FOR RAFLUX
//*   RTCYL         5                  NO. OF CYL. FOR RTFLUX
//*   RTDSP         (,DELETE)          DISPOSITION OF RTFLUX
//*   RTFLUX        &RTFLUX            DSN FOR DATASET RTFLUX
//*   RTVOL         ------             VOLUME FOR RTFLUX
//*   SRFCYL        12                 NO. OF CYL. FOR SURF. FLUXES
//*   XSISO         NULLFILE           DSN FOR DATASET XS.ISO FILE 1
//*   XSISO2        NULLFILE           DSN FOR DATASET XS.ISO FILE 2
//*   XSIVOL        ------             VOLUME FOR XS.ISO FILE 1
//*   XSI2VOL       ------             VOLUME FOR XS.ISO FILE 2
//*
//*   THE FOLLOWING NINE PARAMETERS DEFINE BLOCK ALLOCATIONS
//*   AND DCB'S FOR AUXILIARY FLUX, FDCOEF AND ZONMAP DATASETS
//*
//*   MODEDCB (RECFM=F,BLKSIZE=23220) DCB FOR DIRECT ACCESS FILES
```

Figure 14. IBM Catalogued Procedure APPROC (cont'd.)

```
//*   BLKTYP        CYL              ALLOCATION BY CYLINDERS
//*   SEC           1                SECONDARY ALLOCATION ON DA DATASETS
//*   FDCCYL        20               NO. OF CYL.S FOR FDCOEF DATASET
//*   FLXCYL        1                NO. CYL. FOR 1 GROUP FLUX FILES
//*   PSICYL        5                NO. CYL.S FOR FLUX DATASETS
//*   PSUCYL        3                NO. CYL.S FOR ADJ. UPSCAT. FLUX
//*   ZONCYL        1                NO. CYL.S FOR ZONMAP
//*   CXSCYL        3                NO. CYL.S FOR CXSECT
//*
//*   THE FOLLOWING SIX PARAMETERS DEFINE UNIT AND BLKSIZE FOR
//*   A VARIETY OF DATASETS
//*
//*   HALFTRK       6136             HALF TRACK BLOCKING
//*   QRTRTRK       3064             QUARTER TRACK BLOCKING
//*   TWELTRK       1016             TWELFTH TRACK BLOCKING
//*   UNITS         BATCHDSK         GENERIC UNIT NAME
//*   UNITSCR       SASCR            GENERIC UNIT NAME
//*
//*  ********************************************************************
//*
//APPROC EXEC PGM=&PATH,TIME=&TIMLIM,REGION=&REGN
//STEPLIB  DD DSN=&PRELIB,DISP=SHR
//         DD DSN=&MODLIB1,DISP=SHR
//         DD DSN=&MODLIB2,DISP=SHR
//         DD DSN=&POSTLIB,DISP=SHR
//*
//*            SYSTEM DATASETS
//*
//FT04F001 DD UNIT=SASCR,SPACE=(TRK,(10,10))
//*             DISSPLA SCRATCH FILE
//FT05F001 DD DDNAME=SYSIN
//*             BCD INPUT.
//FT06F001 DD SYSOUT=&DEST,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1596)
//*             PRINTED OUTPUT.
//FT08F001 DD DISP=SHR,DSN=SYS1.DISSPLA.DATA,LABEL=(,,,IN)
//*             DISSPLA FONT FILE
//FT10F001 DD DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1596),SYSOUT=&DEST2
//*             ALTERNATE PRINT FILE.
//FT61 DD DSN=&XSISO,DISP=SHR,UNIT=&UNITS,VOL=SER=&XSIVOL
//*             ARC MICROSCOPIC ISOTOPE CROSS SECTION XS.ISO FILE 1
//FT61F002 DD DSN=&XSISO2,DISP=SHR,UNIT=&UNITS,VOL=SER=&XSI2VOL
//*             ARC MICROSCOPIC ISOTOPE CROSS SECTION XS.ISO FILE 2
//FT62 DD DSN=*.FT61F002,DISP=SHR,VOL=REF=*.FT61F002
//*             ARC MICROSCOPIC ISOTOPE CROSS SECTION XS.ISO FILE 2
//AAFLUX DD DSN=&AAFLUX,DISP=&AADSP,SPACE=(CYL,(&AACYL,1)),
//             UNIT=&UNITS,VOL=SER=&ATVOL,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC ADJOINT ANGULAR FLUX DATASET.
//ADIF3D DD DSN=&&ADIF3D,UNIT=&UNITSCR,SPACE=(TRK,(1,5)),
```

Figure 14. IBM Catalogued Procedure APPROC (cont'd.)

```
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6000)
//*            1,2 OR 3D DIFFUSION MODULE DEPENDENT BCD DATASET.
//ADJANG DD DSN=&ADJANG,UNIT=&UNITS,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            ADJOINT ANGULAR FLUX DATASET.
//ANGSRC DD DSN=&ANGSRC,UNIT=&UNITS,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            ANGULAR SOURCE COMPONENTS.
//AHMG4C DD DSN=&&AHMG4C,UNIT=&UNITSCR,SPACE=(TRK,(1,0)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6000)
//*            CCCC HOMOGENIZATION MODULE DEPENDENT BCD DATASET.
//AISO DD DSN=&&AISO,UNIT=&UNITSCR,SPACE=(CYL,(4,1)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6000)
//*            THE ISOTXS BCD DATASET.
//ALASIP3 DD DSN=&&ALSIP3,UNIT=&UNITSCR,SPACE=(TRK,(3,1)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6000)
//*            THE LASIP-III BCD DATASET
//ANIP3 DD DSN=&&ANIP3,UNIT=&UNITSCR,SPACE=(CYL,(1,1)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6000)
//*            THE ARC SYSTEM GENERAL NEUTRONICS BCD DATASET.
//ARC      DD UNIT=&UNITSCR,SPACE=(CYL,(1,1)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6000)
//*            ARC SYSTEM SPOOLED OUTPUT.
//ASUMMAR DD DSN=&ASUMMAR,UNIT=&UNITSCR,SPACE=(TRK,(3,1)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6000)
//*            BCD INPUT DATASET FOR EDIT MODULE - SUMMARY
//ATFLUX DD DSN=&ATFLUX,DISP=&ATDSP,SPACE=(CYL,(&ATCYL,1)),
//             UNIT=&UNITS,VOL=SER=&ATVOL,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC ADJOINT FLUX INTERFACE DATASET.
//AUDOIT DD DSN=&AUDOIT,UNIT=&UNITSCR,SPACE=(TRK,(3,1)),
//             DCB=(RECFM=VBS,LRECL=84,BLKSIZE=&QRTRTRK)
//*            BCD INPUT DATASET FOR UDOIT MODULES.
//BCDSOB DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//*            BCDSOB PUNCHED OUTPUT FOR LASIP3 CODE
//COMPXS DD DSN=&COMPXS,UNIT=&UNITS,SPACE=(CYL,(3,1)),DISP=&CMPDSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            COMPOSITION MACROSCOPIC CROSS-SECTION DATASET.
//DIF3D DD DSN=&DIF3D,UNIT=&UNITS,SPACE=(TRK,(1,0)),DISP=&DIFDSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&QRTRTRK)
//*            1, 2 OR 3D DIFFUSION MODULE DEPENDENT BINARY DATASET.
//DIRANG DD DSN=&DIRANG,UNIT=&UNITS,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            DIRECT ANGLUAR FLUX DATASET.
//DLAYXS DD DSN=&DLAYXS,UNIT=&UNITS,SPACE=(CYL,(3,1)),DISP=&DLADSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            PRECURSOR YELDS, EMISSION SPECTRA, AND DECAY CONSTANTS
//*            ORDERED BY ISOTOPE
//D3EDIT DD DSN=&D3EDIT,UNIT=&UNITS,SPACE=(CYL,(1,1)),DISP=&D3DSP,
```

Figure 14. IBM Catalogued Procedure APPROC (cont'd.)

```
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            DIF3D EDITS INTERFACE DATASET.
//FIXSRC DD DSN=&FIXSRC,UNIT=&UNITS,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC FIXED SOURCE DATASET.
//FPRINT DD DSN=&FPRINT,UNIT=&UNITSCR,SPACE=(CYL,(1,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            FPRINT FILE FOR LASIP3.
//GEODST DD DSN=&GEODST,UNIT=&UNITS,SPACE=(CYL,(01,1)),DISP=&GEODSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC GEOMETRY DESCRIPTION DATASET.
//GRAPHICS DD PLOTTER=G1DATA,DISP=(MOD,KEEP),DSN=&G1DATA,
//             SPACE=(TRK,(150,10)),UNIT=SASCR
//*            GRAPHICS OUTPUT DATASET
//INPTAP DD DSN=&INPTAP,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            INPTAP FILE FOR LASIP3.
//ISNTXS DD DSN=&&ISNTXS,UNIT=&UNITS,SPACE=(CYL,(04,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC (ISOTXS) TYPE FILE PRODUCED BY CSE010
//ISOTXS DD DSN=&ISOTXS,DISP=&ISODSP,SPACE=(CYL,(&ISOCYL,1)),
//             UNIT=&UNITS,VOL=SER=&ISOVOL,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC NUCLIDE-ORDERED MICROSCOPIC CROSS SECTIONS.
//ISOTXS2 DD DSN=&ISOTX2,UNIT=&UNITS,SPACE=(CYL,(04,0)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC (ISOTXS) FILE USED FOR MERGING IN CSE010
//ISOTXS3 DD DSN=&ISOTX3,UNIT=&UNITS,SPACE=(CYL,(04,0)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC (ISOTXS) FILE USED FOR MERGING IN CSE010
//LABELS DD DSN=&LABELS,UNIT=&UNITS,SPACE=(TRK,(3,0)),DISP=&LABDSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&TWELTRK)
//*            A.NIP3 LABELS AND AREA DEFINITIONS.
//NAFLUX DD DSN=&NAFLUX,DISP=&NADSP,SPACE=(CYL,(&NACYL,1)),
//             UNIT=&UNITS,VOL=SER=&NAVOL,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            RESTART FILE FOR ADJOINT NODAL HEX CALCULATION.
//NDXSRF DD DSN=&NDXSRF,UNIT=&UNITS,SPACE=(TRK,(03,0)),DISP=&NDXDSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC NUCLIDE/CROSS SECTION REFERENCING DATA.
//NHFLUX DD DSN=&NHFLUX,DISP=&NHDSP,SPACE=(CYL,(&NHCYL,1)),
//             UNIT=&UNITS,VOL=SER=&NHVOL,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            RESTART FILE FOR REAL NODAL HEX CALCULATION.
//PKEDIT DD DSN=&PKEDIT,UNIT=&UNITSCR,SPACE=(CYL,(1,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            PEAK POWER DENSITY AND FLUX INTERFACE DATASET.
//PMATRX DD DSN=&PMATRX,UNIT=&UNITS,SPACE=(TRK,(03,0)),DISP=&PMADSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
```

Figure 14. IBM Catalogued Procedure APPROC (cont'd.)

```
//*            CCCC NUCLIDE/CROSS SECTION REFERENCING DATA.
//PWDINT DD DSN=&PWDINT,UNIT=&UNITS,SPACE=(CYL,(01,1)),DISP=&PWDDSP,
//            DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC POWER DENSITY INTERFACE DATASET.
//RAFLUX DD DSN=&RAFLUX,DISP=&RADSP,SPACE=(CYL,(&RACYL,1)),
//            UNIT=&UNITS,VOL=SER=&ATVOL,
//            DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC REAL ANGULAR FLUX DATASET.
//RNDM01 DD DSN=&&PSIOLD,SPACE=(&BLKTYP,(&PSICYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FLUX ITERATE SCRATCH DATASET.
//RNDM02 DD DSN=&&PSINEW,SPACE=(&BLKTYP,(&PSICYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FLUX ITERATE SCRATCH DATASET.
//RNDM03 DD DSN=&&PSIUP,SPACE=(&BLKTYP,(&PSUCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            AUXILIARY FLUX DATASET FOR ADJOINT UPSCATTER ITERATIONS
//RNDM04 DD DSN=&&FDCOEF,SPACE=(&BLKTYP,(&FDCCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FINITE DIFFERENCE COEFFICIENTS SCRATCH DATASET.
//RNDM05 DD DSN=&&FRNOLD,SPACE=(&BLKTYP,(&FLXCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FISSION SOURCE SCRATCH DATASET
//RNDM06 DD DSN=&&FRNNEW,SPACE=(&BLKTYP,(&FLXCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FISSION SOURCE SCRATCH DATASET
//RNDM07 DD DSN=&&FRNM1,SPACE=(&BLKTYP,(&FLXCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FISSION SOURCE SCRATCH DATASET.
//RNDM08 DD DSN=&&FRNM2,SPACE=(&BLKTYP,(&FLXCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FISSION SOURCE SCRATCH DATASET.
//RNDM09 DD DSN=&&SRCNEW,SPACE=(&BLKTYP,(&FLXCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            TOTAL SOURCE SCRATCH DATASET.
//RNDM10 DD DSN=&&ZONMAP,SPACE=(&BLKTYP,(&ZONCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            ZONE MAP SCRATCH DATASET.
//RNDM11 DD DSN=&&CXSECT,SPACE=(&BLKTYP,(&CXSCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            COMPOSITION CROSS SECTIONS SCRATCH DATASET.
//RNDM12 DD DSN=&&FSRC,SPACE=(&BLKTYP,(&PSICYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FIXED SOURCE SCRATCH DATASET.
//RNDM13 DD DSN=&&PSIGO,SPACE=(&BLKTYP,(&FLXCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
//*            FLUX ITERATE SCRATCH DATASET ONE GROUP.
//RNDM14 DD DSN=&&PSIGN,SPACE=(&BLKTYP,(&FLXCYL,1)),
//            DCB=&MODEDCB,UNIT=&UNITSCR
```

Figure 14. IBM Catalogued Procedure APPROC (cont'd.)

```
//*              FLUX ITERATE SCRATCH DATASET ONE GROUP.
//RNDM15 DD DSN=&&RNDM15,SPACE=(&BLKTYP,(1,1)),
//             DCB=&MODEDCB,UNIT=&UNITSCR
//*              FLUX ITERATE SCRATCH DATASET ONE GROUP.
//RTFLUX DD DSN=&RTFLUX,DISP=&RTDSP,SPACE=(CYL,(&RTCYL,1)),
//             UNIT=&UNITS,VOL=SER=&RTVOL,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              CCCC REAL FLUX INTERFACE DATASET.
//RZFLUX DD DSN=&RZFLUX,UNIT=&UNITS,SPACE=(TRK,(01,1)),DISP=&RZDSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              CCCC ZONE AVERAGED FLUX INTERFACE DATASET.
//SCRATH DD DSN=&SCRATH,UNIT=&UNITSCR,SPACE=(CYL,(1,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATH FILE FOR LASIP3.
//SCR001 DD DSN=&SCR001,UNIT=&UNITSCR,SPACE=(CYL,(&SRFCYL,2)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 1.
//SCR002 DD DSN=&SCR002,UNIT=&UNITSCR,SPACE=(CYL,(&SRFCYL,2)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 2.
//SCR003 DD DSN=&SCR003,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 3.
//SCR004 DD DSN=&SCR004,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 4.
//SCR005 DD DSN=&SCR005,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 5.
//SCR006 DD DSN=&SCR006,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 6.
//SCR007 DD DSN=&SCR007,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 7.
//SCR008 DD DSN=&SCR008,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 8.
//SCR009 DD DSN=&SCR009,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 9.
//SCR010 DD DSN=&SCR010,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              SCRATCH FILE 10.
//SEARCH DD DSN=&SEARCH,UNIT=&UNITSCR,SPACE=(TRK,(03,0)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*              CCCC CRITICALITY SEARCH DATA.
//SFEDIT DD DSN=&SFEDIT,UNIT=&UNITS,SPACE=(CYL,(01,1)),DISP=&SFDSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
```

Figure 14. IBM Catalogued Procedure APPROC (cont'd.)

```
//*            SURFACE- AND CELL-AVERAGED POWER DENSITY AND FAST FLUX
//*            DATASET
//SNCONS DD DSN=&SNCONS,UNIT=&UNITSCR,SPACE=(TRK,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC SN CONSTANTS DATASET.
//STACK DD DSN=&STACK,UNIT=&UNITS,SPACE=(CYL,(01,1)),DISP=&STADSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            STACK DATASET
//SYSUDUMP DD SYSOUT=&DMPDEST
//*            SYSTEM DUMP DATASET FOR ABNORMAL NJB TERMINATION.
//UDOIT1 DD DSN=&UDOIT1,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            UDOIT INTERFACE FILE VERSION 1.
//UDOIT2 DD DSN=&UDOIT2,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            UDOIT INTERFACE FILE VERSION 2.
//UDOIT3 DD DSN=&UDOIT3,UNIT=&UNITSCR,SPACE=(CYL,(01,1)),
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            UDOIT INTERFACE FILE VERSION 3.
//ZNATDN DD DSN=&ZNATDN,UNIT=&UNITS,SPACE=(CYL,(01,1)),DISP=&ZNADSP,
//             DCB=(RECFM=VBS,LRECL=X,BLKSIZE=&HALFTRK)
//*            CCCC ZONE NUCLIDE ATOM DENSITIES.
//*  ************************************************************
//*
//*      ANYONE EXPERIENCING DIFFICULTY WITH THIS PROCEDURE CONTACT
//*      C. ADAMS  BLDG 208, ROOM W117, EXT 4820
//*
//*  ************************************************************
//*
```

Figure 14. IBM Catalogued Procedure APPROC (cont'd.)